INFORMATION INTEGRATION FOR CONCURRENT ENGINEERING (IICE) COMPENDIUM OF METHODS REPORT

Richard J. Mayer, Ph.D.

John W. Crump, IV

Ronald Fernandes, Ph.D.

Arthur Keen, Ph.D.

Michael K. Painter

Knowledge Based Systems, Inc.
One KBSI Place
1500 University Drive East
College Station, Texas 77840-2335

HUMAN RESOURCES DIRECTORATE
LOGISTICS RESEARCH DIVISION

2698 G Street

Wright-Patterson Air Force Base, Ohio 45433-7604

JUNE 1995

Interim Technical Paper for Period February 1991 to March 1995

AIR FORCE MATERIEL COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OHIO 45433-7604

INFORMATION INTEGRATION FOR CONCURRENT ENGINEERING (IICE)
COMPENDIUM OF METHODS REPORT

Version 1.0

Richard J. Mayer, Ph.D.
John W. Crump, IV
Ronald Fernandes, Ph.D.
Arthur Keen, Ph.D.
Michael K. Painter

Knowledge Based Systems, Inc.
One KBSI Place
1500 University Drive East
College Station TX 77840-2335
(409) 260-5274

JUNE 1995

Contract No.: F33615-90-C-0012

Prepared for:
Armstrong Laboratory
Logistics Research Division
Wright-Patterson Air Force Base, Ohio 45433-7604
(513) 255-7775

Report Documentation Page

Abstract

For all the rapid advances in computer hardware and specific software technology, enterprise engineering, reengineering, and enterprise integration efforts continue to lack effective, widely understood methods for engineering large-scale information systems. Diverse methods are needed to engineer systems that exhibit desirable life-cycle characteristics (e.g., flexibility, responsiveness, scalability, maintainability, ease of use, integration, performance) and for engaging teams of people in critical life-cycle system development activities. Integration Definition (IDEF) methods, a key product of the IICE program, provide easy-to-use techniques and standard languages of communication that promote good engineering discipline. This report summarizes new IDEF developments toward establishing reliable methods for business constraint discovery (IDEF9), design rationale capture (IDEF6), human-system interaction design (IDEF8), and network design (IDEF14). For each method, the conceptual foundations, relevance, issues, and recommended follow-on development activities are discussed.

Subject terms: Integration, Integration Definition, IDEF, method, methodology, modeling, knowledge engineering, design rationale, constraint, network design, human-system interaction, systems engineering

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

PREFACE

This document provides a summary of research toward the development of four Integration Definition (IDEF) methods: the IDEF9 Business Constraint Discovery method, the IDEF14 Network Design method, the IDEF6 Design Rationale Capture method, and the IDEF8 Human-System Interaction Design method. This work was performed under the Information Integration for Concurrent Engineering (IICE) project, contract # F33615-90-C-0012, funded by the Armstrong Laboratory, Logistics Research Division, Wright-Patterson Air Force Base, Ohio, under the technical direction of United States Air Force Captain JoAnn Sartor and Mr. James McManus. The prime contractor for IICE was Knowledge Based Systems, Inc. (KBSI), College Station, Texas. Dr. Paula S. deWitte was the IICE Project Manager at KBSI. Dr. Richard J. Mayer was the Principal Investigator on this project. Mr. Thomas Blinn was the IICE Technical Manager and also served as the Project Manager during the final close out of this effort. Michael K. Painter was the Methods Engineering thrust manager. The authors gratefully acknowledge the technical support of the Methods Engineering Team whose names are listed below.

Perakath Benjamin, Ph.D.

Bruce E. Caraway

John W. Crump, IV

Ronald Fernandes, Ph.D.

Florence Fillion

Mike Graul, Ph.D.

Umesh Hari

Arthur Keen, Ph.D.

Madhavi Lingineni

Richard J. Mayer, Ph.D.

Christopher P. Menzel, Ph.D.

Michael K. Painter

FOREWORD

Significant technological, economic, and strategic benefits can be attained through the effective capture, control, and management of information and knowledge resources. Like manpower, materials, and machines, information and knowledge assets are recognized as vital resources that can be leveraged to achieve a competitive advantage. The Air Force Information Integration for Concurrent Engineering (IICE) program, sponsored by the Armstrong Laboratory's Logistic Research Division, was established as part of a commitment to further the development of technologies that will enable full use of these resources.

The IICE program was chartered with developing the theoretical foundations, methods, and tools to successfully evolve toward an information-integrated enterprise. These technologies are designed to leverage information and knowledge resources as the key enablers for high quality systems that achieve better performance in terms of life cycle cost and efficiency. The methods research described in this report reflects recent advancements in technology for leveraging available information and knowledge assets.

The name IDEF originates from the Air Force program for Integrated Computer-Aided Manufacturing (ICAM) which developed the first ICAM Definition, or IDEF, methods. Continued development of IDEF technology supports an overall strategy to provide a family of mutually-supportive methods for enterprise integration. More recently, with the expanded focus and use of IDEF methods as part of Concurrent Engineering, Total Quality Management (TQM), and business re-engineering initiatives, the IDEF acronym has been re-cast as an integrated family of Integration Definition methods. Before discussing the development strategy for providing an integrated family of IDEF methods, the components of a method are described in the following paragraphs.

Method Anatomy

A method is an organized, single-purpose discipline or practice.[1] A method may have a formal theoretical foundation, although this is not a requirement. Generally, methods evolve as a distillation of the *best-practice* experience in a particular domain of activity. The term tool is used to refer to a software system that supports the application of a method.

Though a method may be thought of informally as a procedure for doing something plus (perhaps) a representational notation, it may be described more formally as consisting of three components (Figure 1). Each method has (1) a definition, (2) a discipline, and (3) many uses. The method definition is established by characterizing the method's basic motivations, concepts, and theoretical foundations. The definition component is developed by method developers familiar with methods engineering principles (e.g., formal language design, method ontology definition). The discipline component includes the syntax of the method and the procedure by which the method is applied. Many methods have multiple syntaxes which have evolved over time or which emphasize different concepts within the scope of the method.[2] From a method engineer's standpoint, the discipline component is the user interface for the method. This component is often the only one presented to typical users. The use component characterizes how to apply the method in different situations, such as when the method is applied together with other methods versus in a stand-alone fashion.

---

[1] See Coleman, D.S. (1989). *A Framework for Characterizing the Methods and Tools of an Integrated System Engineering Methodology (ISEM), Draft 2, Rev. 0*. Santa Monica, CA: Pacific Information Management, Inc.

[2] Enterprise engineering methods often include a graphical syntax that provides useful visualizations for the information managed by the method. Graphical language facilities serve to document the analysis or design process undertaken and highlight important decisions or relationships that must be considered during method application. The uniformities to which an expert, through experience, has become attuned are thus formally encoded in visualizations that emulate expert sensitivities.

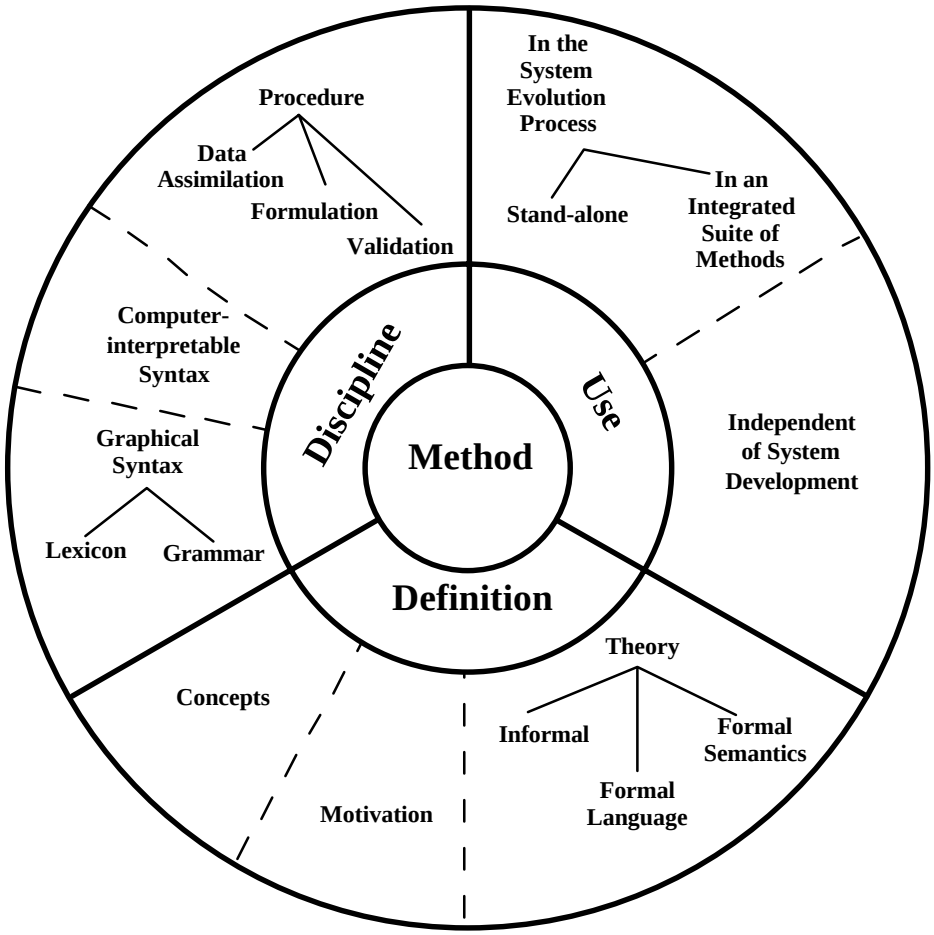


Figure 1.
Anatomy of a Method

Ultimately, methods are designed to facilitate a scientific approach to problem solving. This goal is accomplished by first helping one understand the important objects, relations, and constraints that must be discovered, considered, or decided on. Second, scientific problem solving occurs by guiding the method practitioner through a disciplined approach that is consistent with good-practice experience and leads toward the desired result. Formal methods, then, are specifically designed to raise the performance level (quality and productivity) of the novice practitioner to a level comparable with that of an expert (Mayer, 1987).

Family of Methods

John Zachman, in his pioneering work on information systems architecture, observed:

> [T]here is not an architecture, but a set of architectural representations. One is not right and another wrong. The architectures are different. They are additive, complementary. There are reasons for electing to expend the resources for developing each architectural representation. And, there are risks associated with not developing any one of the architectural representations.

The consistent, reliable creation of correct architectural representations requires the use of a guiding method. These observations underscore the need for *many* "architectural representations," and, correspondingly, many methods.

Methods, and their associated architectural representations, focus on a limited set of system characteristics and ignore those that do not pertain to the task at hand. Methods are not intended to evaluate and represent every possible state or characteristic of the system under study. Hence, the search for a single method, or modeling language, supporting the specification, analysis, design, and representation of all relevant system characteristics continues to frustrate those making the attempt. If such a goal were achievable, the exercise would itself build the actual system, negating the benefits of method application (e.g., problem simplification, low cost, rapid evaluation of anticipated performance, and so forth).

On the other hand, lack of integration among special-purpose methods can be equally frustrating. The IDEF family of methods is intended to strike a balance between special-purpose methods, which are limited to specific problem types, and "super methods" which attempt to include everything. This balance is maintained by providing explicit mechanisms for integrating the results of individual methods within the IDEF family.

Previous research identified critical needs for new methods[3] and led to renewed effort in IDEF method development, with a mandate for compatibility among the family of IDEF methods. New method development has gone in directions where obvious voids existed (rather than reinventing existing methods) with the mission to forge integration links among existing IDEF methods. When applied in a stand-alone fashion, IDEF methods embody knowledge of good practice for the targeted activity. As with any good method, the IDEF methods are designed to raise the performance level of novice practitioners by focusing attention on important decisions while masking irrelevant information and unneeded complexity. Viewed as a toolbox of complementary methods technology, the IDEF family is designed to promote the integration of effort in an environment where effective results have become increasingly dependent on effective use of enterprise information and knowledge assets.

[3] Notably, the Knowledge-Based Integrated Information Systems Engineering Project was conducted at the Massachusetts Institute of Technology (MIT) in 1987, where a collection of highly qualified experts from academic and research organizations, government agencies, computer companies, and other corporations identified method and tool needs for large-scale, heterogeneous, distributed systems integration. See Defense Technical Information Center (DTIC) reports A195851 and A195857.

EXECUTIVE SUMMARY

For all the rapid advances in computer hardware and specific software technology, the bane of enterprise engineering, reengineering, and enterprise integration efforts continue to be the lack effective, widely understood methods for engineering integrated systems. Diverse methods are needed to engineer systems that exhibit desirable life-cycle characteristics (e.g., flexibility, responsiveness, scalability, maintainability, ease of use, integration, performance) and for engaging teams of people in critical life-cycle system development activities.

Methods provide a scientific approach to problem solving. Methods guide their practitioners through a disciplined, reliable approach distilled from expert experience. Methods also highlight important objects, relations, and constraints; and hide irrelevant information and unnecessary detail. Methods are designed to improve performance (quality and productivity) of both individuals and teams involved in systems development activities. Integration Definition (IDEF) methods, a key product of the IICE effort, provide easy-to-use techniques and standard languages of communication that promote good engineering discipline. IDEF methods also improve responsiveness in an environment of rapid and continuous change by helping users to:

- Correctly understand the current environment.
- Propose change.
- Test alternative solutions.
- Predict the impacts of change.
- Successfully implement changes.

IDEF methods facilitate reliable and effective completion of specific tasks in the development process. Individual IDEF methods, when applied in stand-alone fashion, promote consistently good performance of the task for which the method was designed (e.g., information requirements definition, process knowledge capture, object-oriented systems design). The IDEF methods are also designed to work together as a conceptually integrated suite of methods that can interlock like pieces of a puzzle to support the entire development process. Each IDEF method addresses a unique aspect or perspective of enterprise engineering. When individual IDEF methods are applied together, they help achieve one of the key goals of Concurrent Engineering: considering more life-cycle factors early in the design process. The achievement of this goal facilitates the downstream benefits of increased enterprise integration, flexibility, and responsiveness.

The first generation of IDEF methods emerged from the Air Force's Integrated Computer-Aided Manufacturing (ICAM) program in the late seventies. The ICAM program developed the IDEFØ Function Modeling Method, the IDEF1 Information Modeling Method, and the IDEF2 Simulation Modeling Method (See Figure 2).[4] A second ICAM project later developed the IDEF1X Data Modeling Method. IDEF1X facilitates the movement of information requirements, the product of IDEF1 analysis, toward actual systems implementation by establishing a discipline for logical database design.[5] A third generation of IDEF methods emerged from the need for methods technology supporting the development of evolving, information-integrated systems supporting Concurrent Engineering. This development was sponsored by the Air Force Armstrong Laboratory through the Information Integration for Concurrent Engineering (IICE) program and produced methods for process description capture (IDEF3), object-oriented design (IDEF4 and IDEF4 C++), and

---

[4] See SofTech, Inc. (1981). *Integrated Computer-Aided Manufacturing (ICAM): Function Modeling Manual (IDEF0)* (F33615-78-C-5158), Wright-Patterson Air Force Base, OH: Materials Laboratory, Air Force Wright Aeronautical Laboratories.

[5] See D. Appleton Co., Inc. (1985). *Integrated Computer-Aided Manufacturing (ICAM): Information Modeling Manual, IDEF 1—Extended (IDEF1X)* (F33615-80-C-5155), Albany, New York: General Electric Company.

ontology description capture (IDEF5).[6] Preliminary developments were also accomplished through the IICE program toward methods for business constraint discovery (IDEF9), design rationale capture (IDEF6), human-system interaction design (IDEF8), and network design (IDEF14). These partially developed methods are the subject of this report.

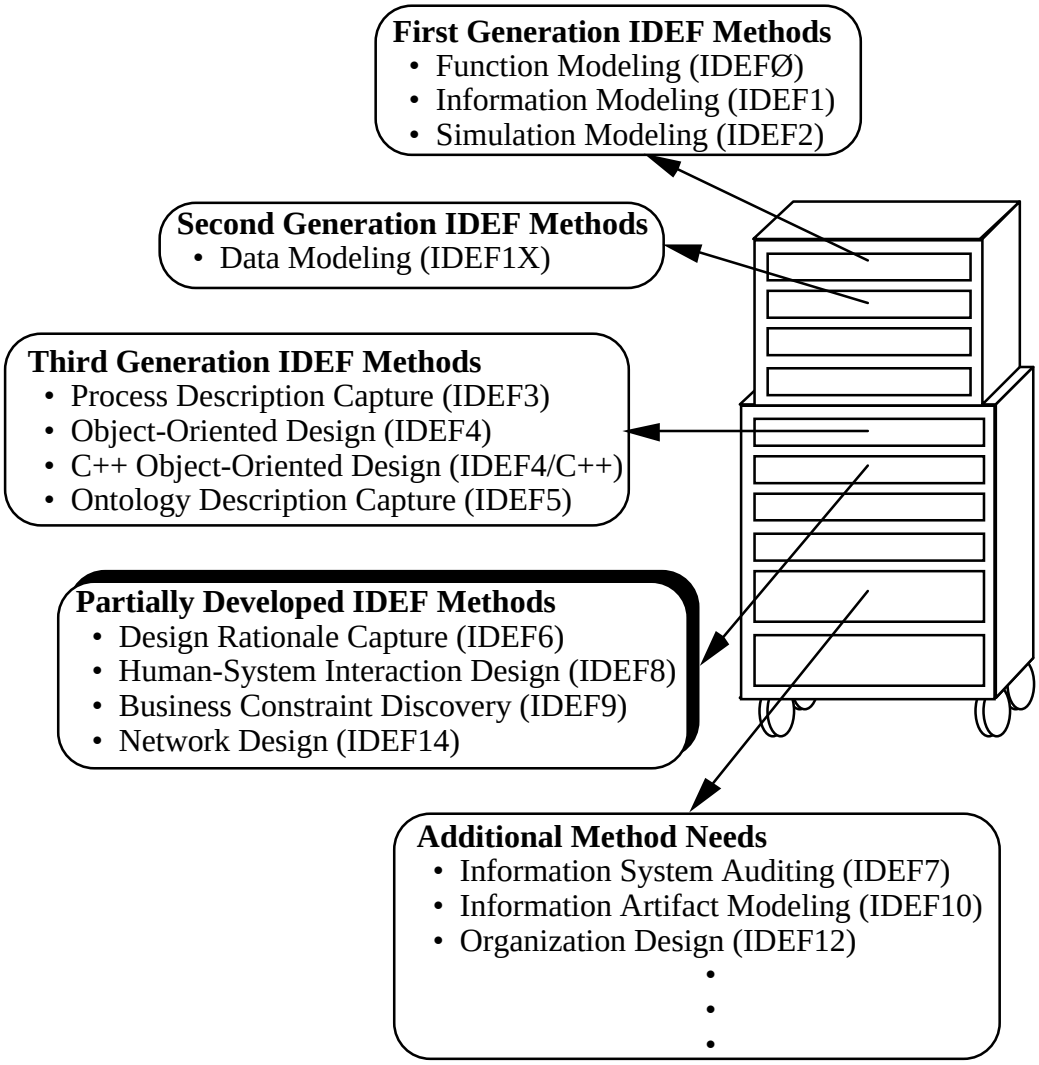


Figure 2.
IDEF Methods: Part of the Systems Engineer's Toolbox

This report describes the conceptual foundations, relevance, issues, and recommended follow-on development activities for each of the methods that were only partially addressed through the IICE program. An introductory description of each method is summarized below in the Executive Summary. More detailed discussions of the research conducted toward the development of each method are found in later sections of the report. The methods are presented in order of relative maturity.

[6] See Mayer, R. J., et al. (1992, 1992, & 1994). *IDEF3 process description capture method report*, *IDEF4 object-oriented design method report*, and *IDEF5 ontology capture method report.* Wright-Patterson Air Force Base, OH: AL/HRGA. Note: New IDEF3 and IDEF4 reports are pending publication.

IDEF9 Business Constraint Discovery Method

Policies, rules, conventions, procedures, contracts, agreements, regulations, and societal and physical laws are the defining structures for an enterprise. These items are the mechanisms for forging relationships between people, information, material, and machines to make a system. In this report, we refer collectively to these items as constraints. If you view an enterprise as a machine, constraints form the architecture and the programming language that define the behavior of that machine. If you view the enterprise as an organism, they form the control structure of that organism, from the genetic code level through the autonomous stimulus response level, to the cognitive behavior level.

The IDEF9 Business Constraint Discovery method described in this report was designed to assist in the discovery and analysis of constraints in a business system. A primary motivation driving the development of IDEF9 was an acknowledgment that the collection of constraints that forge an enterprise system is generally poorly defined. The knowledge of what constraints exist and how those constraints interact is incomplete, disjoint, distributed, and often completely unknown. This situation is not necessarily alarming. Just as living organisms do not need to be aware of the genetic or autonomous constraints that govern certain behaviors, organizations can (and most do) perform well without explicit knowledge of the glue that structures the system. However, if the desire exists to modify the business in a predictable manner, the knowledge of these constraints is as critical as knowledge of genetics is to the genetic engineer.

Constraints are the mechanisms by which humans and nature form systems. Constraints initiate, enable, govern, and limit the behavior of objects and agents to accomplish the goals or purposes of a system. If we want to change the behavior of a system(e.g., improve its performance, efficiency, or effectiveness) we need to know the relevant constraints. The IDEF9 method facilitates the discovery and mapping of the relevant constraints in an organizational system. Once these constraints have been cataloged, they can be systematically examined and, if necessary, tuned or replaced to improve the performance of the system. Constraints often serve a dual role as the glue and the rationale for a system. That is, the collection of relevant constraints often constitutes the description of why the system behaves as it does. From this perspective, IDEF9 provides a reverse engineering tool for the business engineer. It can assist him in discovery of the “logic” behind the design of an existing system. It also provides a mechanism for specifying the logic of a “To-Be” system.

IDEF6 Design Rationale Capture Method

Advancing technology has led to the emergence of products whose expected usable lifetimes extend over decades. Information systems have also evolved from stand alone application-oriented systems with relatively short lifetimes and limited scopes to large scale, distributed systems which must service their users over extended periods of time. Maintenance of information systems whose expected lifetimes may extend over many career periods requires explicit capture and storage of the rationale used in their design.

Design rationale typically exists as unstructured textual comments. In addition to making it difficult to find relevant information on demand, lack of a structured method for organizing and providing completeness criteria for design rationale capture make it unlikely that important information will be documented.

Unlike design methods which serve to document WHAT a design is, new methods are needed to capture WHY a design is the way it is, or WHY it is not manifested in some other form, together with HOW the final design configuration was reached. For the purpose of this discussion, *Design Specification* means capture of WHAT a design is; *Design Rationale* indicates WHY, WHY NOT, and HOW a design arrived at its final configuration; and *Design History* indicates the time-ordered sequence of steps used in the realization of the design.

IDEF6 is intended to be a method with the representational capability to capture information system design rationale and associate that rationale with the design models and documentation for the end system. Thus, IDEF6 attempts to capture the logic underlying the decisions contributing to, or resulting in, the final design. The explicit capture of design rationale serves to help avoid repeating past mistakes, provides a direct means for determining the impact of proposed design changes, forces the explicit statement of goals and assumptions, and aids in the communication of final system specifications. Explicit capture of the motivations for why a designer

selected or adopted a particular design strategy or system feature for enterprise level information systems is essential to the maintenance of that system over its life-cycle.

IDEF14 Network Design Method

IDEF14 is a method that supports the design of computer networks by helping network designers capture requirements, specify network components, capture the existing network configuration, and perform analyses on the design. It also supports managerial decision-making and design to maximize engineering economy. Additionally, IDEF14 represents and stores design rationale of network designs. IDEF14 supports these activities with a tailorable procedure that produces models of network configuration, queuing, reliability, and cost. The network configuration model graphically displays the topology, configuration, specification, and attributes of network components. Using this model, the queuing, reliability, and cost models are generated. These three models are analyzed and used as input to the decision making process of network selection.

IDEF14 provides a reliable procedure that supports data collection, multiple design creation and evaluation, and final design selection. The network design process includes modeling the AS-IS network. Capturing the current and future workloads to assist with formulating TO-BE network designs is also supported and formalized. For each alternative design, multi-faceted analyses are performed, and design rationale is documented. The method also provides for developing and extending libraries of network technology descriptions representing state-of-the-art and future technologies of computer and communication networks. IDEF14 is thus designed to be used together with network simulation tools by assisting with the design of a network architecture that can then be tested and analyzed using varying technology configurations.

IDEF8 Human-System Interaction Design Method

The IDEF8 Human-System Interaction Design Method is used to produce high quality designs of the interactions that do, or should, occur between users and the systems they operate. IDEF8 is not a graphical user interface design method, that is, it is not used to describe screen placement or the size of buttons or windows. It is used to help the system developer capture the interactions that must flow between the system and its users. Systems are collections of objects which perform one or more functions to accomplish a particular goal. The system is not necessarily a computer or a computer program.

Human-system interactions are designed at three levels of specification in the IDEF8 method. The first level defines the philosophy of system operation and produces a set of models and textual descriptions of overall system processes. The second level of design specifies role-centered scenarios of system use. The third level of IDEF8 design is for detailing and refinement of the human-system design. At this level of design, IDEF8 provides a library of *metaphors* used to help users and designers specify the desired behavior in terms of other objects whose behavior is more familiar. Metaphors provide a model of abstract concepts in terms of familiar, concrete objects and experiences. For example, a *light switch metaphor* might be used to specify users interactions involving two possible options. Among the products of this level of design is a human-system interaction mock up with which to test user requirements, formulate user interface strategies (e.g., selecting preferred input and feedback devices), and so forth. Once validated, the products of IDEF8 application are used by system developers (e.g., programmers) to build implementations.

Much of IDEF8's language constructs come directly from the IDEF3 Process Description Capture method because of IDEF8's need for a mechanism to capture and organize process information at multiple levels of abstraction and detail. Specialized language extensions distinguish IDEF8 design models, which are *prescriptive* in nature, from IDEF3 *descriptive* representations.

The fundamental goals of the IDEF8 method are to promote good design practice for human-in-the-loop systems to realize higher quality implementations in less time and at a reasonable cost. IDEF8 seeks to help users produce good human-system interaction designs and consequently higher quality systems by (1) facilitating user-focused data collection, (2) enabling direct user involvement in design activities, (3) focusing efforts on early validation of designs using mock-ups and prototypes, and (4) promoting more productive iterations through the design process.

Summary

IDEF methods provide individuals and teams with a disciplined, reliable approach to problem solving in support of efforts such as enterprise engineering, reengineering, large-scale information systems development, and enterprise integration. The need for multiple, conceptually integrated methods continues to be a growing need that has been partially addressed through the IICE program and earlier initiatives. From these efforts, mature, reliable methods have emerged that are in widespread use throughout government and industry. Additional method needs have been identified, some of which have been investigated through the IICE program. Among these are methods for design rationale capture (IDEF6), human-system interaction design (IDEF8), business constraint discovery (IDEF9), and network design (IDEF14). Continued development is needed to produce mature versions of these additional IDEF methods.

OVERVIEW OF THE METHOD ENGINEERING PROCESS

The approach taken by the IICE methods engineering team to initiate developments toward methods for business constraint discovery (IDEF9), design rationale capture (IDEF6), network design (IDEF14), and human-system interaction design (IDEF8) leveraged knowledge from studying common method engineering practice and experience in developing other analysis and design methods. The purpose of this section is to provide a general overview of that approach.

Figure 3 provides a process-oriented view of the approach used to develop prototype IDEF9 method concepts, a procedure, and candidate graphical and textual language elements. Figure 3 uses the IDEF3 Process Description Capture method[7] to describe this process where boxes with verb phrases represent activities, arrows represent precedence relationships, and "exclusive or" conditions among possible paths are represented by the junction boxes labeled with an "X."

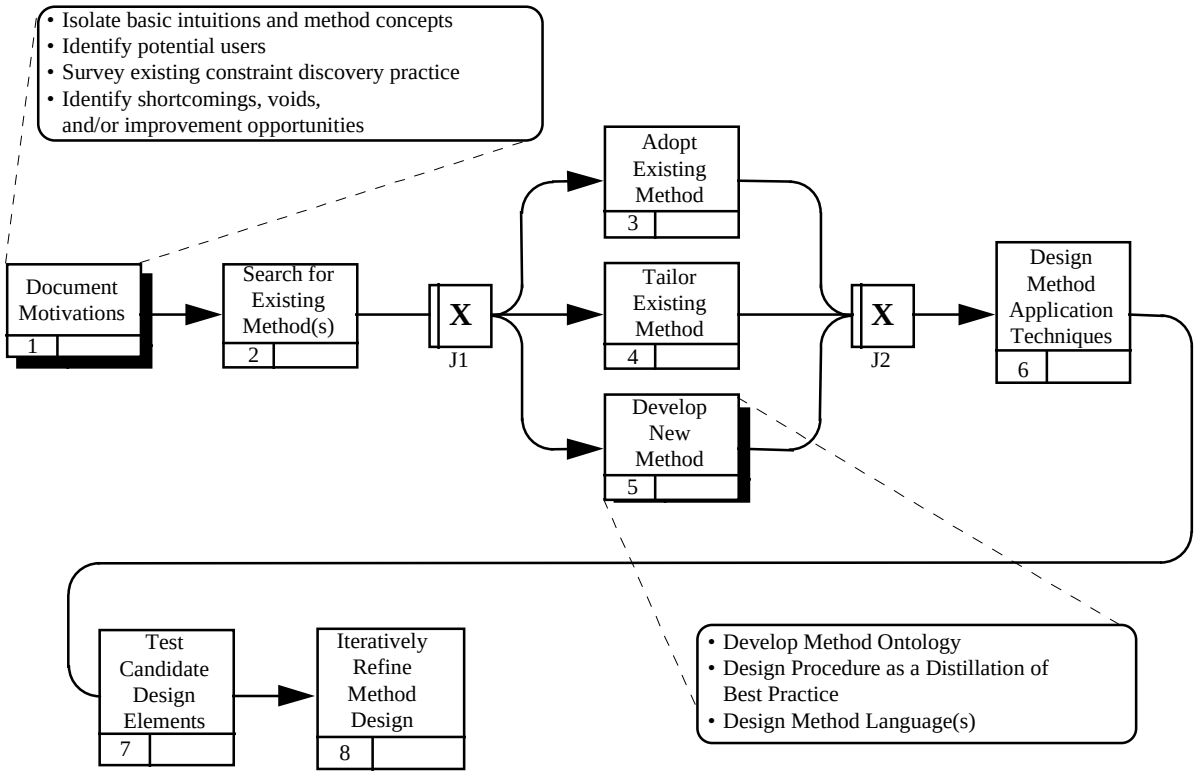


Figure 3.
Process Description of the IDEF9 Development Approach

As is evidenced by Figure 3, one of the basic strategies of methods engineering is reuse. Whenever possible, existing methods are adopted. The next option is to find methods that can satisfy the identified needs with minor modification. This option is an attractive one if the modification does not require a fundamental change in the basic concepts or design goals of the method. Only when neither of these options is viable should method designers seek to develop a new method.

[7] See Mayer, R. J., et al. (1992). *IDEF3 process description capture method report.* Wright-Patterson Air Force Base, OH: AL/HRGA. Note: A new IDEF3 report is pending publication.

A knowledge engineering approach is the predominant mechanism for method enhancement and new method development. In other words, with very few exceptions, method development involves isolating, documenting, and packaging existing practice for a given task in a form that promotes reliable success among practitioners. Expert attunements are first characterized in the form of basic intuitions and method concepts. These are often initially identified through analysis of the techniques, diagrams, and expressions used by experts. These discoveries aid in the search for existing methods that can be leveraged to support novice practitioners in acquiring the same attunements and skills. New method development is accomplished by establishing the scope of the method, refining characterizations of the method concepts and intuitions, designing a procedure that provides both task accomplishment and basic apprenticeship support to novice practitioners, and developing a language(s) of expression. Method application techniques are then developed outlining guidelines for use in a stand-alone mode and in concert with other methods. Each element of the method then undergoes iterative refinement through both laboratory and field testing.

The method language design process is highly iterative and experimental in nature. Unlike procedure development, where a set of heuristics and techniques from existing practice can be identified, merged, and refined, language designers rarely encounter well-developed graphical display or textual information capture mechanisms. When potentially reusable language structures can be found, they are often poorly defined or only partially suited to the needs of the method.

A critical factor in the design of a method language is clearly establishing the purpose and scope of the method. The purpose of the method establishes the needs the method must address. This is used to determine the expressive power required of the supporting language. The scope of the method establishes the range and depth of coverage which must also be established before one can design an appropriate language design strategy. Scope determination also involves deciding what cognitive activities will be supported through method application. For example, language design can be confined to only *display* the final results of method application (as in providing IDEF9 with graphical and textual language facilities that capture the logic and structure of constraints). Alternatively, there may be a need for in-process language support facilitating information *collection* and *analysis*. In those situations, specific language constructs may be designed to help method practitioners organize, classify, and represent information that will later be synthesized into additional representation structures intended for display.

With this foundation, language designers begin the process of deciding what needs to be expressed in the language and how it should be expressed. Language design can begin by developing a textual language capable of representing the full range of information to be addressed. Graphical language structures designed to display select portions of the textual language can then be developed. Alternatively, graphical language structures may evolve prior to, or in parallel with, the development of the textual language. The sequence of these activities largely depends on the degree of understanding of the language requirements held among language developers. These may become clear only after several iterations of both graphical and textual language design.

Graphical language design begins by identifying a preliminary set of schematics and the purpose or goals of each in terms of where and how they will support the method application process. The central item of focus is determined for each schematic. For example, in experimenting with alternative graphical language designs for IDEF9, a *Context Schematic* was envisioned as a mechanism to classify the varying environmental contexts in which constraints may apply. The central focus of this schematic was the *context*. After deciding on the central focus for the schematic, additional information (concepts and relations) that should be captured or conveyed is identified.

Up to this point in the language design process, the primary focus has been on the information that should be displayed in a given schematic to achieve the goals of the schematic. This is where the language designer must determine which items identified for possible inclusion in the schematic are amenable to graphical representation and will serve to keep the user focused on the desired information content. With this general understanding, previously developed graphical language structures are explored to identify potential reuse opportunities. While exploring candidate graphical language designs for emerging IDEF methods, a wide range of diagrams were identified and explored.

Quite often, even some of the central concepts of a method will have no graphical language element in the method. For example, the IDEF1 Information Modeling method includes the notion of an entity but has no

syntactic element for an entity in the graphical language.[8] When the language designer decides that a syntactic element should be included for a method concept, candidate symbols are designed and evaluated.

Throughout the graphical language design process, the language designer applies a number of guiding principles to assist in developing high quality designs. Among these, the language designer avoids overlapping concept classes or poorly defined ones. They also seek to establish intuitive mechanisms to convey the direction for reading the schematics. For example, schematics may be designed to be read from left to right, in a bottom-up fashion, or center-out. The potential for clutter or overwhelmingly large amounts of information on a single schematic is also considered as either condition makes reading and understanding the schematic extremely difficult.

Each candidate design is then tested by developing a wide range of examples to explore the utility of the designs relative to the purpose for each schematic. Initial attempts at method development, and the development of supporting language structures in particular, are usually complicated. With successive iterations on the design, unnecessary and complex language structures are eliminated.[9]

As the graphical language design approaches a level of maturity, attention turns to the textual language. The purposes served by textual languages range from providing a mechanism for expressing information that has explicitly been left out of the graphical language to providing a mechanism for standard data exchange and automated model interpretation. Thus, the textual language supporting the method may be simple and unstructured (in terms of computer interpretability), or it may emerge as a highly structured, and complex language. The purpose of the method largely determines what level of structure will be required of the textual language.

As the method language begins to approach maturity, mathematical formalization techniques are employed so the emerging language has clear syntax and semantics. The method formalization process often helps uncover ambiguities, identify awkward language structures, and streamline the language.

These general activities culminate in a language that helps focus user attention on the information that needs to be discovered, analyzed, transformed, or communicated in the course of accomplishing the task for which the method was designed. Both the procedure and language components of the method also help users develop the necessary skills and attunements required to achieve consistently high quality results for the targeted task.

Once the method has been developed, application techniques will be designed to successfully apply the method in stand-alone mode as well as together with other methods. Application techniques constitute the "use" component of the method which continues to evolve and grow throughout the life of the method. The method procedure, language constructs, and application techniques are reviewed and tested to iteratively refine the method.

Although much progress was made toward defining the IDEF methods described in this report, additional design, testing, analysis, and refinement is needed for each method to achieve full maturity. A summary of the research conducted toward the development of these methods is provided below.

---

[8] The IDEF1X Semantic Data Modeling method uses *'entity'* to describe a different concept than that used in the IDEF1 Information Modeling method. In IDEF1X, an entity is represented explicitly in the method language and represents a set of things that share the same set of attributes.

[9] Language development activity for an IDEF9 method reached the initial levels of this stage of development during the IICE project. Some of the language designs explored during this process are presented later in the report.

TOWARD A BUSINESS CONSTRAINT DISCOVERY METHOD (IDEF9)

Introduction

It is easy to think of instances of policies, rules, laws, or methods that govern the behavior of the components of an enterprise. But, in order to define a method for discovery and analysis of such phenomena, we need to step back and form a perspective for understanding the nature of how these and other constraint forming mechanisms work.

What Is a Constraint?

A *constraint* is a relationship that is maintained or enforced in a given context. The term *relationship* refers to an abstract, general association or connection that holds between two or more conceptual or physical objects. A constraint is simply a special kind of relationship—one that is checked, restricted, or compelled to exist under a given set of conditions. The term *context* refers to such a distinguished set of conditions. A constraint is said to *hold* in a given context when a relationship is maintained or enforced in that context. Conversely, a constraint does not hold in a given context if the constraint is not maintained or enforced for that set of conditions.

Examples are found between objects; between objects and processes: between processes; and between objects and object properties, and the value of those properties. Constraints are expressed in *constraint statements*. For example, a constraint statement expressing a constraint between objects might be "Only project managers are authorized to sign pre-trip requests." An example of a constraint between objects and processes might be: "All travel requires an approved pre-trip request." "Drilling precedes reaming," or "Leasing requires making monthly payments," are statements that "The maximum occupancy of the building complex is two hundred people," or "Product A requires 3600 hours of machining time" are examples of constraint statements involving objects, object properties, and their values.

In this framework, a *system* is characterized as: a collection of objects standing in particular relations and exhibiting particular behavior prescribed by a collection of constraints. In manmade systems, this characterization is usually extended to include the achievement of some goal. It is the behavior and relationship influencing role of the collection of constraints that distinguishes the notion of a system from the more general notion of a situation or state of affairs. Specifically, not all of the properties of, or relationships among, objects in a system are relevant to the system. In fact, particularly in manmade systems, the constraints may specify that particular properties and relations (e.g., equal opportunity constraints) cannot influence the behavior of the system.

Because of their behavior-determining role, constraints have long been the focus of studies directed at understanding and controlling our natural environment. In ecology, for example, constraints between living organisms are studied to understand and maintain the delicate balance of nature. In chemistry, much of the discipline comes from discovering constraints among basic elements and reactive processes. Similar examples could be considered in physics, thermodynamics, medical physiology, and so forth. The study of constraints, however, is not confined to the natural sciences. Similar examples can be found in the study of man-made objects and systems. Constraints manifest in the design process, for example, relations among properties or variables of the proposed artifact and its environment or context [Maher 89]. Design constraints establish the rules, requirements, relations, conventions, and principles that designers must use to synthesize design solutions [Gross 87].

Business systems can be viewed as a collection of objects behaving to perform one or more business functions to accomplish a particular goal under the influence of constraints. Figure 4 illustrates the variety of business systems found in typical manufacturing enterprises. In the analysis of the collection of constraints relevant to a particular business system, it is useful to characterize where a constraint is enforced and controlled relative to a system.

We characterize constraints as: *in* (enforced *in*ternal to the system), *on* (enforced externally on or controlled externally to the system), *of* (possessed by and hence controllable within the system) and *between* (among) (constraints which link business systems as objects together to form larger systems). The performance of a business system, whether operating independently or in concert with other business systems, is governed by constraints.

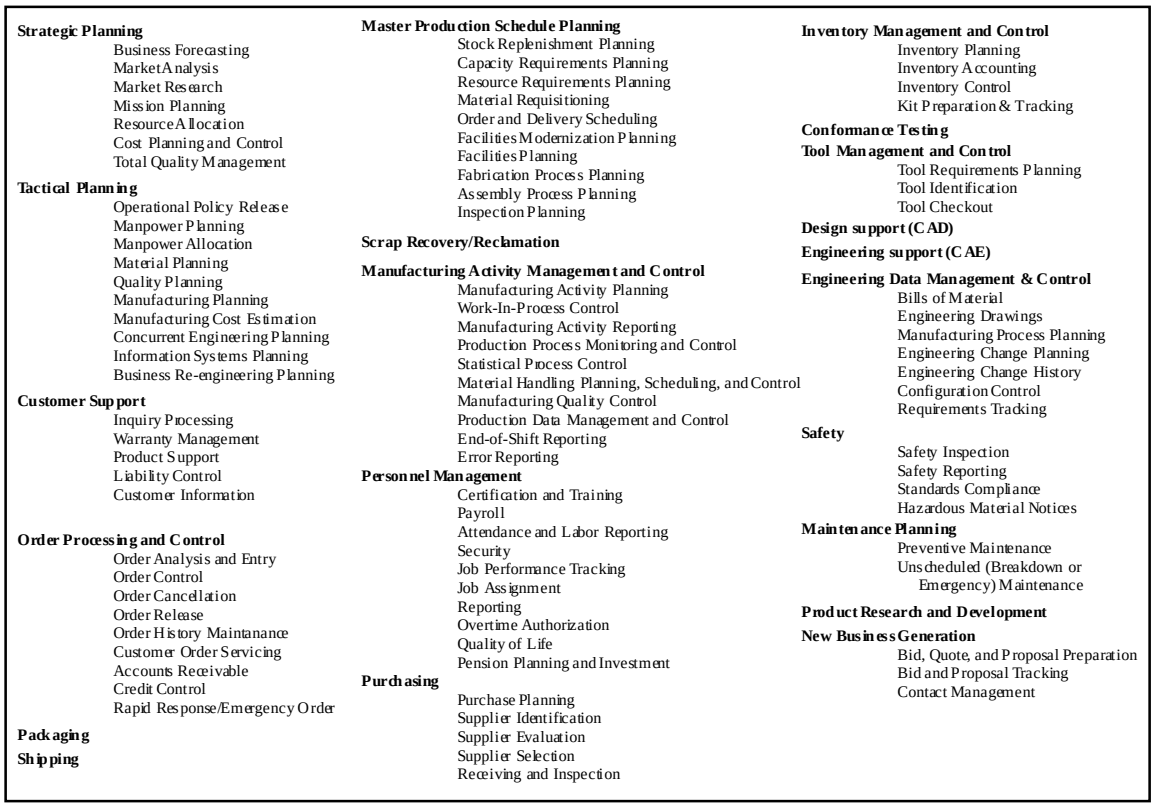


Figure 4.
Typical Business Systems

Constraints can be broadly classified as either *enabling* or *limiting* within a given context (See Figure 5). Although the term "constraint" often evokes images of negative influence or control, constraints serve the vital enabling role of establishing the system. Tolerances between mating parts, for example, establish the constraints required to ensure correct fit. Fiscal management policies and accounting procedures maintained within a company not only keep a pulse on the health of the business but facilitate the prevention of unmanageable debt, fraud, and waste. Both the limiting and enabling aspects or constraints are evidenced in alternative definitions for a constraint in the literature. For example, Eliyahu Goldratt defines a constraint as "anything that limits a system from achieving higher performance versus its goal" [Goldratt]. Other definitions state that constraints are "the rules, requirements, relations, conventions, and principles that define the context of designing [Gross, Ervin, & Anderson, Fleisher]," "specifications, requirements, needs, performance measures, and objectives" [Ullman, Dietrich, and Stauffer], and "a characteristic of the environment, or of the artifact as currently conceived, [that] rules out or against potential settings of design variables" [Smith & Browne]. In other words, whenever the term is used there is an implicit thought that a constraint can be an enabling or limiting factor in design or on performance.

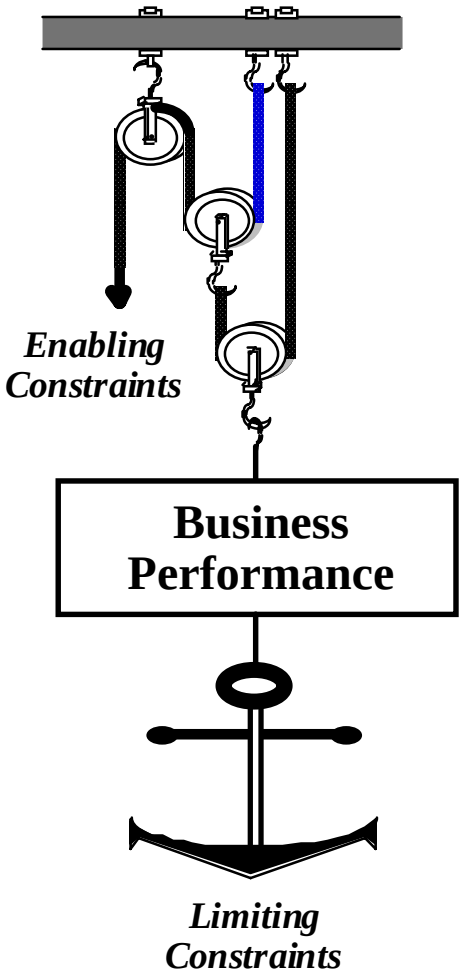


Figure 5.
Constraints Can Be Enabling or Limiting

Limiting constraints are, of course, the most obvious since they tend to manifest themselves through problematic symptoms. Bottlenecks, excessive costs, low quality, long development lead times, waste, and inefficiency are symptoms of limiting constraints. Symptoms are one form of *evidence* that constraints exist within the system. Enabling constraints, such as a constraint for all employees to begin work no later than 8:00 AM, constitute relationships maintained to promote good business system performance. Evidence supporting the existence of these constraints may also be found.

We define *evidence* as an indication, sign, or manifestation that supports or proves the existence of a constraint in a given context. Evidence of constraints can take many forms. Some of these include symptoms (observable evidence of a system failing to meet goals), operating instructions, procedure manuals, employee handbooks, regulations, specifications, policy manuals, project files, design models, and so forth. These are not the constraints themselves but an indication, sign, or manifestation of possible constraints. Policy statements written in a policy manual, for example, would support the proposition that there are constraints whose description is found in the policy manual. If the policies are not maintained or enforced, however, no constraint exists.

The existence of a constraint implies the existence of a system that maintains or enforces the relationship. One should be cautious, however, not to confuse a constraint with the system that maintains the constraint. A constraint is simply a special kind of association between a relationship that is maintained and the system that maintains that relationship.

However, it should be obvious that because constraints involve the use of a system in their maintenance or enforcement, constraints come at a cost. The maintenance of natural constraints is reflected in terms of energy expended or transformed. Business constraints are maintained by business systems, and operating business systems costs time and money.

Motivation for Collecting and Managing Business Constraints

Given the influence constraints have on business performance, one would expect organizations to spend as much time and effort on managing constraints as on managing the objects that comprise business systems. Evidence of constraint management activity is abundant, although it may not be recognized. Authority is given or denied, policies are changed and assignments made, standards are developed and enforced, and so forth. However, decision-makers and system developers often have limited support to identify and manage constraints effectively.

Effective constraint visibility and management enables decision-makers to recognize and respond appropriately to constraint problems typical of many businesses, as illustrated in Table 1.

Table 1. Some Typical Problems

Type I problem	The cost of maintaining a constraint exceeds the value of the constraint.
Type II problem	Constraints exist that no longer support the organization's goals.
Type III problem	The constraint causes unintended or undesirable effects.
Type IV problem	The agent or system (mechanism) responsible for maintaining the constraint fails to consistently or correctly enforce the constraint.
Type V problem	What was believed or intended to be a constraint lacks any explicit maintenance or enforcement mechanism.

Without visibility of constraints and support for constraint management, business owners, strategic and tactical planners, and process owners fall prey to constraints that limit performance and frustrate attempts to seize opportunity. Systems overburdened with outdated or inappropriate constraints unnecessarily levy costs in overall business performance and consume resources that could be used elsewhere. According to Eliyahu Goldratt, a widely recognized proponent of constraint-driven change, outdated rules, policies, and procedures are the main cause of limited organizational performance [Goldratt & Cox 86]. In such an environment, decisions-take a reactive stance toward constraints. Problematic symptoms and limiting constraints take center stage, demanding most, if not all, of the decision-maker's attention. Lacking visibility on the current constraints of the enterprise, decisions are made without the benefit of tools to anticipate downstream effects. Increasingly, decisions made to solve problems at a local level create unanticipated negative effects on the overall enterprise. Opportunities also take a back row seat as resources and time become increasingly scarce.

Interestingly, while many negative symptoms are most effectively dealt with by changing or eliminating existing constraints, many decision-makers instead add new business constraints. This tendency may actually exacerbate the challenge of business constraint identification and management. Informal mechanisms of business constraint management thus tend to grow increasingly inadequate as the organization adapts to changes in the business environment, as the number and scope of business systems increase, and as the corporate memory of constraint development history fades with time.

Benefits of Constraint Identification

Constraints initiate, enable, govern, and limit the behavior of objects and agents to accomplish the goals of a system. If we want to change the behavior of a system for what ever reason (e.g., improve its performance, efficiency, or effectiveness) we need to know the relevant constraints. Business constraint identification and management provide decision-makers with increased visibility of the business constraints that govern achievable performance. Discovering, cataloging, and maintaining business constraints thus enables decision-makers to deal more effectively with constraint-related problems (See Table 2).

The ability to catalog business constraints can assist decision-makers in designing and prioritizing constraints relative to organizational goals. Knowledge of interrelationships among constraints also enables more reliable performance predictions and change impact assessments. By identifying and eliminating unnecessary constraints, costs are eliminated, performance improvements (e.g., schedule and quality gains) can be realized, and freed resources can be used to leverage new opportunities. Often these benefits can be realized without any additional investment in automation or information systems.

Table 2. Some Benefits of Constraint Discovery

Constraint-related problem	Benefit of constraint discovery
The cost of maintaining a constraint exceeds the value of the constraint.	Enable decision-makers to identify and eliminate constraints that exceed the value they provide.
Constraints exist that no longer support organizational goals.	Enable decision-makers to identify and eliminate outdated or unnecessary constraints.
The constraint causes unintended or undesirable effects.	Enable decision makers to reengineer or eliminate constraints that produce unintended or undesirable effects.
The agent or system (mechanism) responsible for maintaining the constraint fails to consistently or correctly enforce the constraint.	Enable systems developers to identify and remedy system designs that fail to appropriately enforce constraints.
What was believed or intended to be a constraint lacks any explicit maintenance or enforcement mechanism.	Enable decision-makers to identify missing systems needed to maintain both precautionary and enabling constraints.

Somewhat less obvious, perhaps, is the fact that knowledge of constraints can yield new sources of information and expose misinformation. For example, knowledge of the constraint that trees add one ring each year to their circumference, when coupled with knowledge of the number of rings in a given tree, yields new information—the age of the tree. Similarly, knowledge of business constraints can be used to yield otherwise inaccessible information about the health and productivity of the business, to determine how quickly products can be produced, to estimate what it takes competitors to produce their products, and to identify where changes can be made to achieve competitive advantage.

Motivation for a Method to Collect Constraints

While it is easy to see the value of managing business constraints, it is not so easy to discover and catalog them. For one thing, it is common for people to confuse constraints with policy statements, symptoms, or the objects that maintain a constraint. Furthermore, like a river flowing over a riverbed filled with rocks of various shapes and sizes, constraints often are not apparent until they become a visible obstacle in the path of workflow. Recognizing obvious obstacles in the path is only part of the constraint discovery undertaking. Less obvious obstacles, such as those beneath the surface of a river, must also be identified if safe passage is to be achieved. Knowledge of constraints can be used as an extension of power. They also provide a framework that bolsters a sense of security within the workplace. These factors can result in reluctance or even animosity for any initiative to surface and eliminate unnecessary constraints.

The IDEF9 method for constraint discovery provides a systematic approach for business owners, strategic and tactical planners, systems developers, project leaders, and decision-makers to identify and document business constraints. A number of methods provide partial support for constraint discovery. Goldratt's theory of constraints, Quality Function Deployment (QFD), Taguchi's influence diagrams, the IDEF1 Information Modeling method, and others address different aspects of constraint discovery and documentation. There are few methods, however, that explicitly distinguish between simple relations and constraints. Often, when constraints are captured in a given method the range of constraints captured is confined to a restricted set of constraint types (e.g., precedence constraints in IDEF3). Additionally, there are no methods that provide a systematic approach to discover, document, validate, and refine both enabling and limiting business constraints.

Once constraints in an organization have been cataloged they can be systematically examined adjusted, eliminated to improve the performance of the system. Constraints often serve a dual role as both the glue and the rationale for a system. That is, the collection of relevant constraints often constitutes the description of why

the system behaves as it does. From this perspective, IDEF9 provides a reverse engineering tool for the business engineer. It can assist him in discovering the "logic" behind the design of an existing system or for specifying the logic of a "To-Be" system.

The IDEF9 method is intended to guide practitioners in rapidly and reliably discovering, displaying, characterizing, and validating business constraints. Given the wide range of anticipated users, we have designed the prototype IDEF9 method to be easily used by personnel who represent all segments of the business.

Users and Key Beneficiaries of IDEF9

Two broad categories of users for an IDEF9 Business Constraint Discovery method can be considered. First, direct users will apply the method to identify, document, validate, and refine the corporate library of constraints. These users range from those who manage constraint discovery efforts to those who perform the detailed evidence collection and analysis. The direct users of IDEF9 include managers of enterprise improvement initiatives (e.g., Total Quality Management, Business Reengineering), strategic and tactical planners, knowledge workers, and systems developers (both internal and external to the organization). Strategic and tactical planners will gain tremendous leverage through the use of IDEF9 as critical assumptions are questioned, external and internal forces are identified and analyzed, future challenges and opportunities are predicted, and requirements for supporting business systems are developed. System developers are supported with a mechanism to discover and refine constraints through specification and design. The second class of users, such as business owners and managers, do not necessarily make direct use of the method, but use the products of an IDEF9 application effort. For them, the most visible element of an IDEF9 Business Constraint Discovery method is not the set of techniques used to extract, validate, and refine constraints from the domain but the graphical language facilities used to display constraints. These users will apply the knowledge captured in constraint libraries to identify patterns, perform change impact assessments, and identify new avenues of potential improvement.

Potential Applications for IDEF9

The IDEF9 Business Constraint Discovery method was designed to assist in the discovery and analysis of constraints in an enterprise. The collection of constraints that forges an enterprise system is generally poorly defined. That is to say, the knowledge of what constraints exists and how those constraints interact is at best incomplete, disjoint, distributed, and often times completely unknown. This situation is not necessarily alarming, just as a living organism need not be aware of the genetic or autonomous constraints that govern certain behaviors, an organization can (and most do) perform well without explicit knowledge of the glue that structures the system. However, if the desire exists to improve business performance or adapt to changes in a predictable manner, then knowledge of these constraints is critical.

Knowledge of business constraints can benefit efforts like Business Process Reengineering (BPR), Total Quality Management (TQM), strategic planning, constraint-driven information systems design, Activity Based Costing (ABC), and so forth. Each of these efforts aims to improve organization performance by breaking away from outdated rules governing how we organize and conduct business. Recognizing the constraints of the business is the first step to finding imaginative new ways to conduct business. Once those constraints have been recognized, outdated constraints can be eliminated. An on-line library of constraints, indexed to display the constraints relevant to a given business context (e.g., doing business with DoD, doing business in the semiconductor manufacturing industry) could provide additional support for leveraging new opportunities. Using the knowledge contained in an on-line library of constraints, business owners and system developers could explore opportunities to expand into new areas of business and rapidly determine the costs of maintaining the constraints necessary to operate in those environments.

Summary of Developments and Research Findings

IDEF9 Basic Concepts

A thorough understanding of IDEF9's basic concepts is needed to effectively apply the procedural and language components of the prototype method. Among these are the following concepts:

Constraint	Context
Evidence	Effect(s) of a constraint
Symptom	System
Rationale for the constraint	Goal

This section describes these concepts and provides examples of each.

Constraint

In the IDEF9 method, a constraint is defined as a relationship that is maintained or enforced in a given context. Policies, rules, conventions, procedures, contracts, agreements, regulations, and societal and physical laws maintained within the enterprise establish the defining structure for the enterprise. These items are the mechanisms for forging relationships between people, information, material, and machines to make a system. If you view an enterprise as a machine, constraints form the architecture and the programming language that define the behavior of that machine. If you view the enterprise as an organism, they form the control structure of that organism, from the genetic code level through the autonomous stimulus response level, to the cognitive behavior level.

Constraints are expressed in *constraint statements*. Examples of constraint statements include:

1. Protect against liability.
2. Minimize in-process inventory.
3. Maximize cost recovery.
4. Load first-class passengers, families with small children, and disabled passengers before all others.
5. Load the aircraft beginning with passengers seated in rear.
6. All projects produce a final report.
7. Projects whose contract value exceeds $10M require an additional cost report.
8. An individual must maintain a current license to operate a vehicle.
9. Only cleared personnel may enter the secure area.
10. A group leader must be a member of the management team.

Notice that use of the imperative form is not necessarily the only form that a constraint statement might take. Furthermore, finding a statement that uses the imperative form does not necessarily ensure that one has found a constraint. For example, constraint #9 above may have appeared as the command "Challenge all unidentified personnel within the secure area."

Throughout the process of constraint discovery, analysts identify *candidate constraints*, or relations suspected to be ones that are maintained or enforced in a given context. Candidate constraints are first substantiated by the analyst, who examines the data. Candidate constraints are then challenged by domain experts. This process eventually leads to a refined collection of validated constraints and context characterizations establishing the conditions in which the constraints hold.

Context

The term *context* refers to a distinguished set of conditions. Each context in a constraint model must be unique. A *context label* is a short descriptive phrase (e.g., when removing asbestos, procuring computer hardware, [being] at the construction site) used to convey a general understanding of the context boundaries. A more in-depth look at the context is generally needed to distinguish one context from another. In other words, a label alone may not suffice in distinguishing one context from another. For this reason, those applying IDEF9 should catalog the minimal set of *essential* facts or conditions that uniquely distinguish one context from another. *Accidental* facts and conditions may also be noted. However, those facts or conditions that must hold (i.e., essential facts or conditions) define the minimal set needed to bind the context.

Evidence

We define *evidence* as an indication, sign, or manifestation that supports or proves the existence of a constraint in a given context. Just as chemical reactions are manifested by the resultant by-products, the existence of a constraint will manifest itself through some form of evidence. For example, a candidate constraint statement such as "All purchase requests require project manager approval and senior company official authorization" might be hypothesized from the presence of one signature block for the project manager and another for an authorization official on the company's Purchase Request form. The Purchase Request form is said to be evidence supporting the proposition that there is a constraint requiring project manager approval and senior company official authorization on all purchase requests. Evidence must be grounded in the appropriate context and validated before the existence of the constraint can be established. For example, an authorization signature may only be required for purchases made using government contract funds (e.g., computer hardware hardware and software purchases). Overhead projects, on the other hand, may require only project manager approval to purchase the necessary material.

The most easily identified evidence of a constraint is the existence of a problematic symptom. Such a condition may indicate the existence of a limiting constraint, the lack of an enabling constraint, or both. For example, excess in-process inventory and bottlenecks at various points in the production line may indicate constraints that maximize the efficiency of individual processes or the lack of constraints on raw material ordering. Limiting capacity constraints might be identified by finding the places in the production line where bottlenecks occur. An enabling constraint may be identified by noting that rates in a production line are set to match that of the slowest process. Sources of evidence also include documents outlining policies, procedures, requirements, designs, and so forth. Forms with signature blocks, operating instructions, procedure manuals, handbooks, regulations, standards, specifications, policy manuals, project files, and design models also represent possible evidence.

Effects of a Constraint

The term *effect* is used to describe something that inevitably follows an antecedent (as a cause or agent). Constraints initiate, enable, govern, and limit the behavior of objects and agents in the system. These behaviors are generally referred to as *effects*. Constraints also establish cause-effect chains that propagate effects through the system. Thus, effects are often considered either *direct* or *indirect*. Effects may also be *intended* or *unintended*, and *desirable* or *undesirable* in a given context. For example, a manufacturing company may have a constraint to collect metrics reflecting the productivity of individual shops. Presumably, this constraint would provide decision-makers with the visibility needed to identify where excess capacity exists and where additional resources are needed. This constraint may give rise to an unintended, direct effect wherein shop foremen create work for their people to keep them busy, which inflates the data reviewed by decision-makers. An indirect effect is the creation of excess in-process inventory and downstream bottlenecks.

Symptom

Symptoms indicate a condition which impairs correct functioning of a system. Symptoms provide subjective evidence of a condition that is used to aid in correctly diagnosing the condition. That is, the same symptoms may indicate one or more conditions impairing normal functioning. A fever, for example, may indicate the presence of an infection or the flu. In business systems, the lack of some core competency may give rise to poor quality products, late delivery, and so forth. While symptoms are observed failings of a system, they are often confused with *concerns* which are "possible" failings of a system.

Although symptoms are not the condition themselves, they are often problematic. Domain experts often refer to symptoms as *problems*. *Problem*, however, often connotes the *source* of distress or difficulty and may therefore lead to the wrong conclusions. The IDEF9 constraint discovery process leverages domain expert attunements to symptoms, probable causes, and effects to assist users in identifying candidate constraints or the lack of needed constraints. Symptoms are used to hypothesize cause-effect constraints relative to other symptoms. For example, a production supervisor may describe frequent failures of Integrated Circuits among the key problems that they are attempting to address. The production workforce, however, may view those failures as a symptom of high turnover rates, contamination, improper handling, insufficient training, and so forth. Manufacturing engineers may attribute the failures to poor moisture control, cracked dies, or broken and bent leads. Design engineers may attribute failures to oxide or silicon defects. Each of these observations reflects knowledge of constraining relations which may also be symptoms of other conditions.

System

A *system* is a collection of objects standing in particular relations and exhibiting particular behavior prescribed by a collection of constraints. In manmade systems, this characterization usually includes the notion of the achievement of a goal. *Business systems* can be viewed as a collection of objects behaving to perform one or more business functions under the influence of constraints to accomplish a particular goal. Several examples of business systems comprised of multiple objects are provided in Figure 4 above.

The systems of primary interest for IDEF9 are systems that maintain or enforce a given constraint or set of constraints. Examples of a system that maintains or enforces a constraint might include individual objects (e.g., particularly an active object or agent such as an account manager or an authorization official) or collections of objects (e.g., an accounting system or a prime contractor).

Rationale for the Constraint

The *rationale* of a constraint is the set of beliefs motivating the establishment and maintenance of a constraining relationship. This set of beliefs includes those held to be true when the constraint exists as well as those held to be true in situations where the constraint does not exist. The rationale of a constraint is documented to assist with periodic review of the currency and relevance of the constraint.

Goal

A *goal* is defined as an object or end that one strives to achieve. Business systems exist to accomplish a particular goal or set of goals. Goals may stand in a number of relationships with other goals. Among these are *depends-on-existentially, implies, is-part-of,* and so forth. Ideally, each goal in the business will be oriented to contribute to an overall set of goals. Goals, however, are highly dependent on the environment and are subject to frequent changes and reprioritization. Changes in the organization's goals motivate changes to the constraint set used to direct the organization toward achieving those goals.

IDEF9 Procedure Developments[10]

This section presents a prototype procedure for constraint discovery, validation, and refinement. The procedure presented in this section assumes a large constraint discovery effort involving a team approach. Projects that are narrower in scope may not require all these activities. As with all methods, the application procedure depends largely on the purpose for which the method is being used. Those undertaking a constraint discovery project are therefore encouraged to prepare a detailed method application guide at the beginning of the project.

Constraint discovery is an evolutionary process through which candidate constraints are identified, validated, and refined. In general, when using IDEF9 to discover and document constraints, the following six steps are applied recursively:

1. Collect - Acquire observations and sources of evidence for constraints.

2. Classify - Individuate contexts, objects, object types, properties, and relations.

3. Hypothesize - Postulate candidate constraints from the data and evidence acquired.

4. Substantiate - Generate or collect examples to determine which candidate constraints should be promoted to the status of a constraint.

5. Challenge - Involve domain experts in testing the validity of analysts' conclusions.

6. Refine - Filtering, improving, adjusting, and adding detail to constraint characterizations.

These steps are embodied in the prototype IDEF9 procedure presented below. The activities comprising IDEF9's procedure should be considered "modes of thought" rather than sequential steps. Users should not expect to apply these activities in a strictly sequential manner, or that organizing activities by project phases necessarily defines when those activities start or stop. Rather, phases reflect which modes should or do predominate during a given interval of time. Thus, modes of activity may be organized into phases to assist with management of the project. The following section provides a functional description of the modes of activity constituting IDEF9's procedure, thus establishing a basic framework for constraint discovery.

Mode Zero: Define the Project

The constraint discovery team must establish the purpose and scope[11] of the constraint discovery effort as early as possible in the project. The purpose statement provides a "completion criteria" for the constraint discovery effort. The purpose is usually established by prioritizing objective statements for the effort, stating the requirements of the constraint discovery effort and examining questions or findings that the client wants answered. The scope of the project is established by a set of statements that bound or delimit the area of the domain addressed by the project. Scope statements identify the specifically targeted areas of constraint discovery activity and identify those areas that are explicitly ignored.

The purpose and scope can rarely be determined completely and accurately in advance. The client often revises their list of needed findings or questions as the data is compiled. The area an analyst thinks will lead to the answer often leads to areas that were not considered within the scope. The purpose and scope generally

---

[10] Significant reuse of the procedural components from the IDEF3 Process Description Capture and IDEF5 Ontology Description Capture methods facilitated the development of the prototype procedure description that follows.

[11] One of the central concepts in constraint discovery is the notion of a context in which a constraint holds. A different meaning for *context* is generally applied among IDEF method practitioners using other IDEF methods. *Context* is used with other IDEF methods to describe the scope or boundary of the project. To avoid unnecessary confusion, *scope* has been adopted for IDEF9 when describing the boundaries of the project.

evolve during the initial part of the project. The purpose and scope of an IDEF9 effort are captured on an IDEF9 Project Summary Form similar to the one shown in Figure 6.

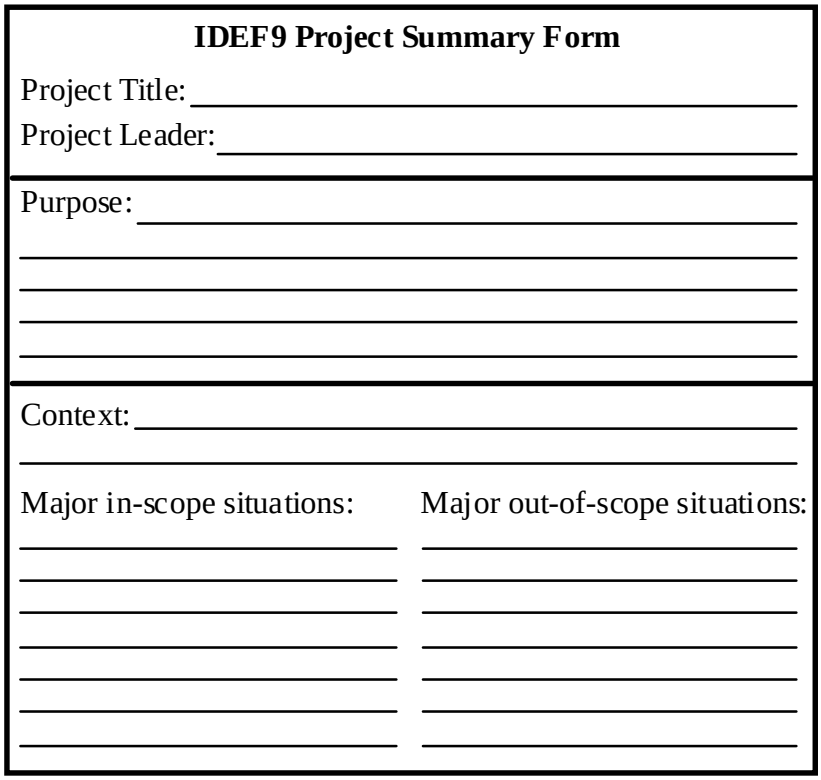


Figure 6.
IDEF9 Project Summary Form

Define the Purpose

Defining the purpose is an important initial step in the constraint discovery effort. If the purpose is taken for granted or ignored, project personnel are likely to find the results of their efforts ignored by or of little use to the client. Without a purpose statement, the only completion criteria are budget and time. Conversely, with a regularly reviewed and clearly defined purpose, the project can often be completed within budget. Defining the purpose involves listing the stated objectives of the client and the specific source(s) of each (e.g., person, project, or organization), defining the information goals of the project in terms of how the constraint information will be used, and establishing priorities among the stated objectives and information goals of the effort. The process of developing a purpose statement can be facilitated by involving the client in answering questions like the following:

1. What problematic symptoms, concerns, or opportunities are of the greatest interest to the client?

2. Who will use the constraint information once it is available?

3. What question(s) does the client need answered?

4. What issues are behind the need for constraint discovery?

5. What decisions are behind the need to identify constraints?

Establish the Scope

Once the purpose of the effort has been characterized, it is possible to define the scope of the project. Defining the scope of the project begins with setting the boundaries of the constraint discovery effort and documenting those boundaries in a set of scope statements. Ideally, scope definition should identify only areas that are relevant to the needs of the client.

An effective mechanism for defining the scope of the project is identifying the situation types or contexts to be considered and identifying those that fall outside the project boundaries. Characterizing the situation types of interest may begin at a course-grained level by developing a descriptive phrase (such as with an adverb phrase like *working* for government agencies, *disposing* of hazardous materials) and a brief description for the contexts of interest. Characterizing the contexts of interest involves achieving a consensus among constraint discovery team members on the title and paragraph description for the contexts. It is common for differently named contexts to be nearly identical. Conversely, it is also common for different contexts to be named the same. The similarity or dissimilarity among contexts will initially become evident through the development of paragraph descriptions. Consensus among team members may require more fine-grained definition of the contexts, particularly as the team members review the purpose and scope statements periodically through the project. When necessary, more detailed characterizations of the contexts of interest can be developed by identifying the participating objects, relations, and facts that must hold in the contexts. Additionally, those contexts that impact or are directly related to the in-scope contexts, but which are outside the scope of the project, should be identified. Those intimately familiar with the domain must be relied upon to actually identify the contexts.

Scope and level of detail decisions are tentative at this stage of the project and should be updated as the constraint data becomes available. An astute project leader will regularly assess the adequacy of the constraint data captured with respect to the specified needs and information goals of the client.

Mode One: Organize for Data Collection

Once the initial project purpose and context have been determined, the task of organizing for data collection can begin in earnest. At this point, the makeup of the project team will be solidified, team member roles will be established, and scenario development responsibilities will be assigned to team members.

The following roles are normally assumed by personnel involved in a constraint discovery effort.

1. Analyst: The IDEF9 expert who will be the primary developer of the IDEF9 constraint models.

2. Client: The person or organization requesting the constraint discovery effort development.

3. Domain expert: The person possessing direct knowledge about the domain of interest.

4. Primary contact: The individual who acts as the interface between the analyst and the domain expert.

5. Project leader: The person ultimately responsible for the entire constraint discovery effort.

6. Reviewers: Persons knowledgeable in the domain and/or the IDEF9 method responsible for reviewing and approving draft models and documents. Reviewers authorized to make written critiques of IDEF9 schematics are *commentors*. The remainder are *readers*. Both team members and domain experts can be reviewers.

7. Librarian: A person assigned the responsibility of maintaining source material logs and files of documents, making copies, distributing kits, and keeping records.

8. Team members: All personnel involved with the IDEF9 constraint discovery effort.

For large projects, the role of the librarian is essential. In smaller efforts, that role may be assumed by the analyst. In establishing the librarian function, the project leader assigns an individual to be responsible for collecting, cataloging, controlling, and distributing source material, kits, glossaries, files, and so forth throughout the project. Additionally, the librarian is responsible for assembling reference models and materials from external sources that can be used to accelerate team efforts. The librarian may also maintain a glossary of terms as a reference to be used during interviews to ensure that analysts understand terminology that is unique to a discipline, industry sector, company, or company segment. Whether maintained by the librarian, or informally shared among analysts, the glossary of terms will grow and undergo incremental refinement throughout the project.

A pivotal task in organizing the data collection effort is identifying the key sources of knowledge and information about the domain. Working with the primary contact, the project leader or analyst compiles a list of experts to be interviewed. In compiling this list, it is helpful to obtain background information about each expert. This includes information about the responsibilities, current assignments, and other areas within or related to the domain in which the expert has experience. The name, location, and telephone number of the experts should also be recorded.

Throughout the data collection effort, other valuable sources of information will be sought and identified. Some of these include operating instructions, procedure manuals, employee handbooks, regulations, policy manuals, project files, reusable IDEF models, and models derived through the use of other methods and techniques. These items often constitute evidence of constraints themselves or provide references to evidence in the domain.

In addition to organizing the structure of the team, the project leader also needs to organize the activities of the team. Organizing the constraint discovery activity may begin by casting the general IDEF9 procedure into a more formalized method application guide tailored to the specific needs of the project. A method application guide outlines a project-specific application of the IDEF9 procedure tailored to meet the needs of the effort. Among the items that may be included in the method application guide are modeling conventions to be used, standard outlines for interviewing domain experts, method and tool interface specifications, project library use procedures, and a standard glossary of terms. This guide may be accompanied by a project plan. A typical project plan will delineate phases of effort with clearly established tasks and milestones, intermediate and final deliverables, individual team member assignments, informal and formal reporting structures, and so forth.

Mode Two: Collect and Analyze Evidence

Having organized the team and outlined the approach, the team will begin constraint discovery by engaging in evidence collection. Constraint discovery team members document expert observations and collect evidence of constraints by direct interaction with domain experts. This data is later analyzed to form the basis for hypothesizing constraints.

Prepare for Interviews

The most valuable mechanism of evidence collection is the interview. Interviews with domain experts afford the interviewer an opportunity to collect special insights, both into normal situations and the exceptions to the normal situations within the domain. Direct observation techniques allow the interviewer to observe normal situations and are often used to augment interviews with the domain expert.

While the specific interviewing approach and format are likely to vary across projects, some guidelines are recommended. Before the interview, the analyst should prepare a tentative agenda and some specific questions. Analysts are encouraged to prepare a brief outline of the purpose of the interview, the topics to be covered in the interview, the types of information being sought, the authority for requesting the interview, and probing questions that can be used to motivate discussion. On large projects, project leaders may wish to include more formalized interview preparation guidelines and standards in a method application guide—including standard interview planning sheets, question templates, glossaries of terms, and so forth.

The ultimate success of the interview depends largely on the preparation made by the analyst. A number of activities contribute to successful preparation:

1. Schedule the interview and make necessary logistics preparations.

2. Establish the goal(s) of the interview.

3. Prepare candidate questions.

4. Anticipate the probable questions and concerns of the person being interviewed and be prepared to resolve concerns.

Once a list of experts to be interviewed has been compiled, an interview schedule can be developed. Interviews are normally scheduled with domain experts through the primary contact. The analyst should make sure that the scheduled time and duration of the interview is coordinated with the person being interviewed and his or her supervisor. Additional logistics considerations are also important to the success of the interview, such as reserving a suitable location to conduct the interview and arranging for the necessary supplies. Analysts also generally find it useful to plan the attire they wear to the interview in order to convey a professional appearance and still set the interviewee at ease.

The goal(s) of the interview should also be established up front. In establishing the interview goals, analysts establish why the interview is being scheduled and what information is needed from the domain expert. Preparing a succinct goal statement often helps to provide a general direction for the interview line of questioning.

Once the goal(s) of the interview has been established, candidate questions can be formulated. Candidate questions should be written down and organized into a logical sequence. Candidate questions should be clear, use words and phrases appropriate to the background of the person being interviewed, and invite rather than lead answers. In preparing candidate questions, it is often useful to explore the following topics:

1. What are the organization's goals and objectives?

2. What are the organization's Critical Success Factors (CSFs) and performance measures?

3. What are the organization's problematic symptoms?

The answers to these questions often provide valuable guidance in identifying business constraints. Statements of goals and objectives give strong indications of perceived environmental constraints and, with appropriate follow-through, can lead to the discovery of deeply-rooted belief systems and undocumented constraints. Quite often, constraints discovered through lines of questioning centered around organization goals will reveal enabling constraints and constraints that no longer support current goals. When this line of questioning is applied across different organization levels, hidden transformations between strategic and tactical goal structures may be revealed that can be used to identify missing and/or inappropriate constraints. In a similar fashion, exploration of the organization's CSFs and performance measures yields important constraint information. In fact, there is not likely to be any more obvious evidence of existing constraints than artifacts of performance measurement (e.g., graphs, charts, reports). Constraints arising from performance measures, while generally intended to be enabling, often drive unintended and undesirable behavior. Finally, valuable guidance in discovering constraints may be obtained through listing problematic symptoms and the influencing factors believed to be the underlying cause(s) of those symptoms. Symptoms of problems may be manifest in business systems as bottlenecks, excessively long cycle times, poor quality, high cost, and so forth.

In preparing candidate questions for the interview, analysts should be cautious not to over prepare. The exercise of writing down questions and analyzing the way they are formed serves to build good interviewing skills. The time invested to this activity must be weighed against the possibility that the questions may not be used. Questions may be eliminated through the discovery of new information or to follow a line of discussion that was not previously anticipated.

An element of preparation often overlooked by inexperienced analysts is the need to explain why the domain experts are being interviewed, what will be done with the information they provide, and what they can expect in return. Each interview should establish a mutual understanding of these items before attempting to satisfy the information needs of the analyst. The following list is representative of the topics and concerns that the analyst should be prepared to address [Harrington 91].

1. Why the interview is being conducted.

2. Who authorized the interview

3. Who else is being interviewed.

4. How the interviewee was selected and by whom.

5. How the information will be used.

6. Whether the person will be anonymous.

7. Whether the person will be quoted in summary findings.

8. What feedback the person will receive.

9. How the person might participate in the outcome of the process.

10. What is in it for the interviewee.

11. Why highly detailed, accurate information is important to the success of the interview and the project.

12. How the person plays a key role in an important process.

Interview Domain Experts

Interviews may be conducted at any time throughout the project with one or more of the following goals in mind:

1. To collect additional information.

2. To confirm and/or clarify previously collected information.

3. To validate candidate constraints with the domain expert.

4. To obtain leads for acquiring additional information.

Interviews with domain experts are critical. The analyst (interviewer) should create a positive and friendly atmosphere during the interview. The interviewer should attempt to convey to the domain expert the feeling that they are working together to discover constraints and solve some problem for the organization. A novice interviewer should constantly remind himself that the expert is the one with the knowledge of how a organization works. Generally, the expert is interested in helping and will often provide questions and lines of investigation that the interviewer had not thought of pursuing. In constraint discovery, this is the bonus for good preparation.

Domain experts often begin by describing rules, policies, procedures, and relations that should be maintained and those that actually are. Questions that help to distinguish desired operating conditions from the norm, and normal operating conditions from work-arounds or special cases can help guide the interview. The main focus of the interview should be on rules, policies, procedures, and relations that are currently maintained

or enforced (i.e., constraints), rather than "Should-Be" conditions that may not be maintained. When focused on constraints, analysts should also be cautious to avoid talking about "To-Be" constraints to avoid introducing bias in the domain expert's answers. Throughout the interview, constraint information provided by the domain expert needs to be faithfully recorded in a form that can be shared among all team members. Analysts should pay particular attention to the use of the imperative form in the description or in documents provided by the domain expert (e.g., *Complete* the attached job application form). Words like *must*, *will*, *shall*, *always*, and *never* are often included in imperative phrases (e.g., Applicants must complete the attached job application form before they can become eligible for an interview). However, neither the absence nor presence of these terms necessarily indicates a constraint. Logical quantifiers like *all*, *every*, *some*, and *none* also provide clues for discovering candidate constraints.

Collect and Catalog Evidence

As appropriate, analysts should request copies of artifacts that constitute forms of evidence of constraints. Evidence of business constraints can take many forms including procedure manuals, instruction sheets, forms with fields for approval signatures, handbooks, policy manuals, regulations, specification documents, standards documents, strategic and tactical plans, organization charters, mission statements, efficiency reports, and so forth.

Other sources of evidence include analysis models (e.g., IDEFØ function models, IDEF1 information models, IDEF3 process descriptions, IDEF5 ontology descriptions) and design models (e.g., IDEF1X semantic data models, IDEF4 object-oriented design models) that are relevant to the project.

The IDEFØ function modeling method captures some constraint-related information, although at a relatively course-grained level. For example, objects modeled as controls on a particular activity often list artifacts describing how the activity is or should be performed (e.g., documents containing information about rules, policies, and procedures). Objects classified as mechanisms represent the means by which the activity is accomplished, thus providing valuable assistance in cataloging and validating constraints. IDEFØ controls below the artifact level must be validated to determine whether they represent constraints, since IDEFØ does not explicitly capture which mechanisms enforce which controls.

IDEF1 information models capture and display a specialized class of constraints; specifically, those constraints that are maintained in the domain through the information system. In other words, IDEF1 is used to model constraints for which some information objects have been designed and implemented. Similarly, IDEF3 process descriptions explicitly capture another specialized class of constraints. IDEF3 captures precedence and causality relations among processes and events within the environment. IDEF3 also captures constraints relative to the state change behavior among objects participating in a process. Ontologies developed using IDEF5 include characterizations of objects, object properties, and relations, thus providing a solid foundation for constraint discovery. IDEF5 ontologies also distinguish between defining versus non-defining and essential versus accidental properties and relations. Accessibility to IDEF5 descriptions can therefore greatly accelerate the process of constraint discovery. Design models (e.g., IDEF1X and IDEF4 models) also capture reusable constraint information. IDEF1X models capture the design constraints to which information system developers must conform. These constraints reflect business rules and policies to be implemented through the information system. IDEF4 models capture similar information with a specific target toward implementation in an object-oriented language.

All data that is collected during the course of the project should be logged on an IDEF9 Evidence Log as illustrated in Figure 7.

USED AT:	ANALYST: I.M. Modeler DATE: 28 Feb 93 PROJECT: IICE NOTES: 1 2 3 4 5 6 7 8 9 10 REV:	X	WORKING DRAFT RECOMMENDED PUBLICATION	READER: DATE:	CONTEXT:

Evidence Log #	Evidence Name/Description	Received From	Comments
EL#1	Purchase Requisition/Form PI-R6 4-72	U.R. Buyer	
EL#2	Procedure #079-003 /Rev. 00 "Preparation of the Requisition"	U.R. Buyer	
EL#3	Procedure #079-001/ Rev. 00 "Preparation of the Purchase Order"	Policy and Procedures Manual	
EL#4	Procedure #101-506 "Purchasing Codes"	Policy and Procedures Manual	
EL			
EL			
EL			
EL			
EL			
EL			
EL			

TITLE: Evidence Log	NUMBER:

Figure 7
Evidence Log

Analyze Collected Data

Following data collection, interview notes are compiled, the Evidence Log is updated to reflect newly collected evidence, and the initial findings are cataloged into lists called pools. Among the pools found to be potentially useful in organizing and analyzing constraint-related information are the following:

1. Business goals pool
2. Performance measure pool
3. Symptom pool
4. Source statement pool
5. Context or situation type pool
6. System or object pool
7. System or object property pool
8. Relation pool.

In analyzing the collected data, analysts may also perform the following activities:

- Further refine and individuate contexts (e.g., working for the Air Force, [being] at the construction site, acquiring replacement parts, [conducting] Programmed Depot Maintenance [PDM]).

- Associate existing business goals with context(s).
- Associate existing performance measures with context(s).
- Associate existing symptoms with context(s).
- Catalog the objects involved with context(s).
- Catalog object properties.
- Identify relations.

A focus on relations can be of great assistance in uncovering candidate constraints during analysis. Relations may be found between contexts, between objects and contexts, between object types and object instances, between object types and property values, and so forth. Table 3 below illustrates candidate constraint statements illustrating different kinds of relations.

Table 3. Different Relations Illustrated by Constraint Statements

	Context	**System**	**Property**
Context	• Working government contracts involves maintaining an auditable accounting system. • Drilling precedes reaming.	• Carry a current driver's license while operating a motor vehicle. • People are not allowed on the construction site without a hard hat.	• Children under thirteen are to be accompanied by a parent when attending PG-13 rated movies.
System		• People own cars. • John owns the Le Baron. • The tensile strength of steel is greater than that of iron.	• Birds have feathers. • Bill's hourly wage is $7.25. • First-line supervisors sign off all flight safety-critical maintenance operations.
Property			• The number of rings in a living tree corresponds directly to its age in years.

It is also often useful to classify constraints to assist in discovery and downstream reuse of constraint information. Two criteria should be considered when developing a classification or taxonomy of constraints. First, the taxonomy should express *orthogonality* among categories, i.e., each category of the taxonomy should be disjoint such that every element in the taxonomy's domain can be uniquely assigned. Second, the taxonomy should be *exhaustive* over the specific domain.

Constraint granularity and coupling among constraints should also be considered when developing a taxonomy. Granularity is the level of abstraction used to represent the constraint. The more abstract the representation, the more difficult it is to assign a constraint to a unique category. For example, one of the constraints used in production management is "in-process inventory between stations should be kept balanced." The level of granularity represented by this constraint statement is very high. There are many factors that contribute to a balanced production line, e.g., machine capabilities, down-time, set-up time, material routing

strategies, and so forth. More fine-grained constraint statements such as "the work in process (WIP) for station A is 4 units or less" and "WIP for station B is under 10 units" permit unique classification to a subcategory of balanced production constraints called "WIP limits." Coupling among constraints may also need to be considered. That is, constraints often stand in relationships that make it difficult to divide them into separate categories. Again, refining the granularity of the constraint representation often helps one to effectively categorize constraints.

Several possible classifications for constraints may be considered to systematically analyze candidate constraints and validated constraints. For example one might find it useful to classify constraints in terms of the measure of control that the organization has over the constraint's structure or very existence. Constraints could then be divided between those that are *volitional* (imposed by choice) and those that are *non-volitional* (no choice) in a given context. Constraints may also be classified as *enabling* or *limiting* relative to some goal in a given context. Constraints dealing with resources may broadly be categorized as *resource constraints*; those dealing with the capacity properties of systems as *capacity constraints*; those arising from the selection of one design strategy versus another as *design constraints*; those for which the rationale is largely unknown or poorly justified as *status quo constraints*; and so on.

The IDEF9 method does not prescribe one classification scheme or set of classification schemes over any others. The most appropriate classification scheme(s) will be determined by how the constraint information needs to be used. It is generally useful, however, to adopt several classifications to permit analysis of the constraints within and across those classifications. This analysis often leads to the discovery of new constraints and to previously unrecognized opportunities for improvement.

Mode Three: Hypothesize Candidate Constraints

Using source statements and the evidence collected, members of the constraint discovery team hypothesize *candidate constraints*. A candidate constraint can be thought of as a "first pass" at a constraint. A candidate constraint that can be supported by data collected is said to be *substantiated*. A candidate constraint is said to *mature* into a *constraint* upon successfully passing further validation testing.

Common ways in which candidate constraints emerge are as follows:

1. Candidate constraints are obvious to the modeler can be substantiated based on the evidence collected.

2. Constraints that emerge due to personal belief systems of the domain experts can be substantiated based on interview notes or the evidence collected.

3. Constraints that emerge from characteristic functions (e.g., height of a table) and can be substantiated by the data collected.

4. Candidate constraints that are suspected by the modeler but which cannot be supported by the evidence collected so far can be hypothesized and later substantiated (or unsubstantiated) through additional evidence collection.

For each candidate constraint identified, the following information will be recorded on a candidate constraint specification form:

1. Candidate Constraint ID#

2. Constraint statement

3. Constraint description.

4. Context ID#(s) (Context(s) in which the constraint holds)

5. Arguments of the candidate constraint

6. System or object(s) that maintain(s) the constraint

7. Constraint violation consequences

8. List of supporting evidence.

Mode Four: Validate and Refine Constraints

Candidate constraints must undergo a validation process to ensure that they are in fact constraints. The validation process can be divided into two parts, both of which are necessary. The first element of validation is characterized by analysts attempting to *substantiate* candidate constraints using evidence, interview notes, and direct observations collected by the team. The second element of validation involves engaging domain experts in challenging the candidate constraints that remain. Throughout the validation process, candidate constraints and constraints undergo a refinement process wherein derivative versions of constraint statements are proposed and tested, pool data is extended, and more in-depth characterizations are developed.

Substantiate Candidate Constraints

Once candidate constraints have been hypothesized, analysts attempt to substantiate their hypotheses. In effect, analysts set out to prove that they have actually discovered constraints. Substantiating candidate constraints involves testing them against example instances of contexts to determine whether the relations thought to be constraining are in fact maintained. Any one of the following situations may arise while attempting to substantiate candidate constraints:

1. The candidate constraint is substantiated.

2. The candidate constraint is accepted after refining the proposed characterization of the context(s) in which the constraint holds.

3. The candidate constraint can be substantiated with slight modification.

4. Both candidate constraint and the context(s) in which it holds undergo slight modification to support substantiation.

5. The hypothesis of a candidate constraint is found to be unsubstantiated.

When discrepancies surface, analysts may need to refine their characterization of the holding contexts to establish the validity of the constraint based on currently available evidence. Alternatively, they may need to refine their characterization of the candidate constraint. Either situation generally requires additional data. A number of approaches are available for collecting additional information. These include:

1. Conducting follow-up interviews to answer questions and/or identify additional evidence.

2. Arranging for direct observation of the situation(s) included in the scope of the effort.

3. Revisiting source material with a new focus of analysis.

4. Conducting facilitated workshops.

The approach or combination of approaches will be determined by both the nature of the information needed and the purpose for which IDEF9 is being used. Any additional data collection activity will involve making appropriate updates to previously collected data (e.g., updating the evidence log).

Candidate constraints having undergone this step in the analysis are migrated to an intermediate classification as either *substantiated* and *unsubstantiated*.

Challenge Candidate Constraints

Both substantiated and unsubstantiated candidate constraints are subject to domain expert review and validation. Thus, domain experts *challenge* the analyst's conclusions. If the constraint discovery team has a strong body of evidence to justify its hypothesis, there is high probability that substantiated candidate constraints will be promoted to the status of a constraint. On occasion, unsubstantiated constraints will also be supported by new evidence provided by the domain expert at this stage of the process. The various steps involved in the validation of a constraint are:

1. The constraint discovery team provides domain experts with a list of substantiated and unsubstantiated candidate constraints and the supporting evidence for their hypothesis.

2. The constraint discovery team interacts with domain experts to obtain and record feedback.

3. The constraint discovery team analyzes feedback obtained from the domain experts.

4. The constraint discovery team refines validated constraints and their associated context descriptions based on the acquired feedback.

Refine Constraints

Refinement is a process of filtering, improving, and adding value to a product. The process of constraint discovery is itself a refinement process. Hence, refining constraints is an ongoing activity that occurs throughout the constraint discovery effort. More precisely, the constraints themselves are not refined. Rather, the characterization of those constraints is refined. The extent of refinement is largely determined by the purpose of the project, although companies interested in maintaining libraries of constraints will want to adopt standards for the information to be managed about business constraints. The following guidelines aid in a full characterization of the discovered constraints:

1. Identify correlations between contexts and business constraints that hold in those contexts. Contexts can be classified using any number of classification schemes. It is often most useful, however, to classify contexts based on the degree to which they share constraints.

2. Identify correlations between business constraints and the system(s) or object(s) responsible for maintaining the constraints. Because business constraints are defined as relations that are maintained or enforced in a given context, it is important to identify the object(s) responsible for maintaining business constraints. Knowledge of the object(s) that maintain constraints is useful in helping identify and resolve resource contention problems; determining the impact of absent individuals, systems, and processes; and exploring alternative mechanisms to maintain a desired constraint.

3. Document the motivation(s) for the business constraint. The motivation of a constraint characterizes the justifications, intuitions, assumptions, and judgments giving rise to its existence. Among the assumptions that should be captured are the presumed consequences of the constraint not being in place. This will help to identify those constraints that no longer need to be maintained.

4. Identify correlations between business constraints and organization goals. Both positive and negative correlations can be established between business constraints and the goals of the organization for a given context. When analyzing constraints across contexts, analysts may find constraints that support one organization goal while conflicting with others. Correlations between constraints and goals enable downstream analysis of the impact of constraints on organization goals, as well as providing support for sensitivity, cause-effect, and influence analyses. Prioritization of constraints can also be performed for a given business situation.

5. Document correlations between constraints, performance measures, and effects. Performance measures are among the most obvious evidence of candidate business constraints. Performance measures often serve to drive behavior patterns within the company, at times in ways that were not previously anticipated or desired. When undesirable effects surface, it is often valuable to revisit both the performance measures and the constraints which provide management visibility on those performance measures.

6. Correlate observed effects (intended and unintended) and symptoms with business constraints. Cause-effect chains can be established by linking constraint interrelationships across contexts. Documenting the effects of a constraint helps establish these correlations.

7. Further classify constraints using classification schemes that are likely to provide the greatest downstream value to the organization. Several potentially useful classification schemes are presented in the *Basic Concepts* section.

IDEF9 Language Design Developments

A number of candidate schematic types were explored to support the constraint discovery process. The general questions used to guide the development of candidate schematics are as follows:

1. What step(s) of the procedure is a schematic needed or desired?

2. What information (type, amount, etc.) should the schematic convey?

3. How is the information to be conveyed? What will be the view (perspective) adopted? What graphical metaphor will be used (hierarchical, spider's web, network, sequential, etc.)

4. What is the syntax for the graphical language? List the different elements to be represented and the graphical symbol used for each element.

5. What is the semantics of each graphical construct?

In answer to these questions, six candidate schematics have been identified, subject to further testing and analysis. Among these were the following:

1. Context schematic

2. Constraint resource schematic

3. Constraint-relationship schematic

4. Constraint effects schematic

5. Goal schematic

6. Goal-relationship schematic

7. Symptom schematic

Each of the candidate schematics explored is described in the following sections.

Context Schematic

Purpose: The purpose for this schematic is to help users (1) establish and incrementally refine the scope of a constraint discovery effort, (2) display the constraints that hold in a given situation, and (3) rapidly identify shared constraints among distinguished contexts.

Viewpoint: Context-centered

Procedure component(s) supported:

- Mode Zero, Establish project context
- Mode Two, Collect and analyze evidence
- Mode Three, Hypothesize candidate constraints
- Mode Four, Validate and refine constraints

Graphical metaphor: Hierarchical

Candidate syntax:

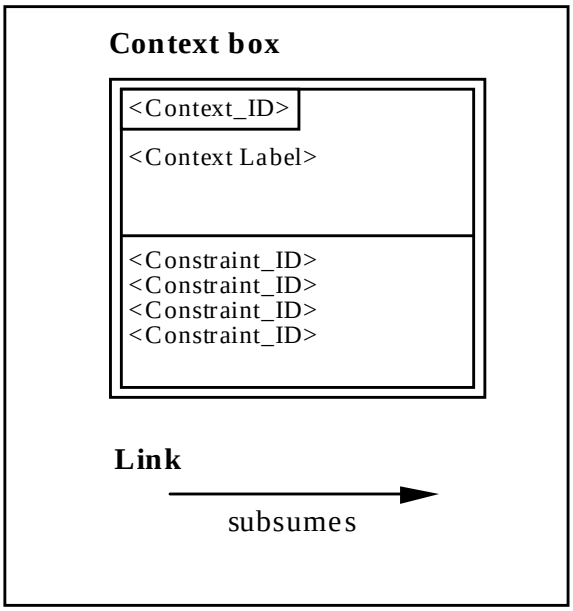


Figure 8.
Candidate Syntax for the Context Schematic

Example:

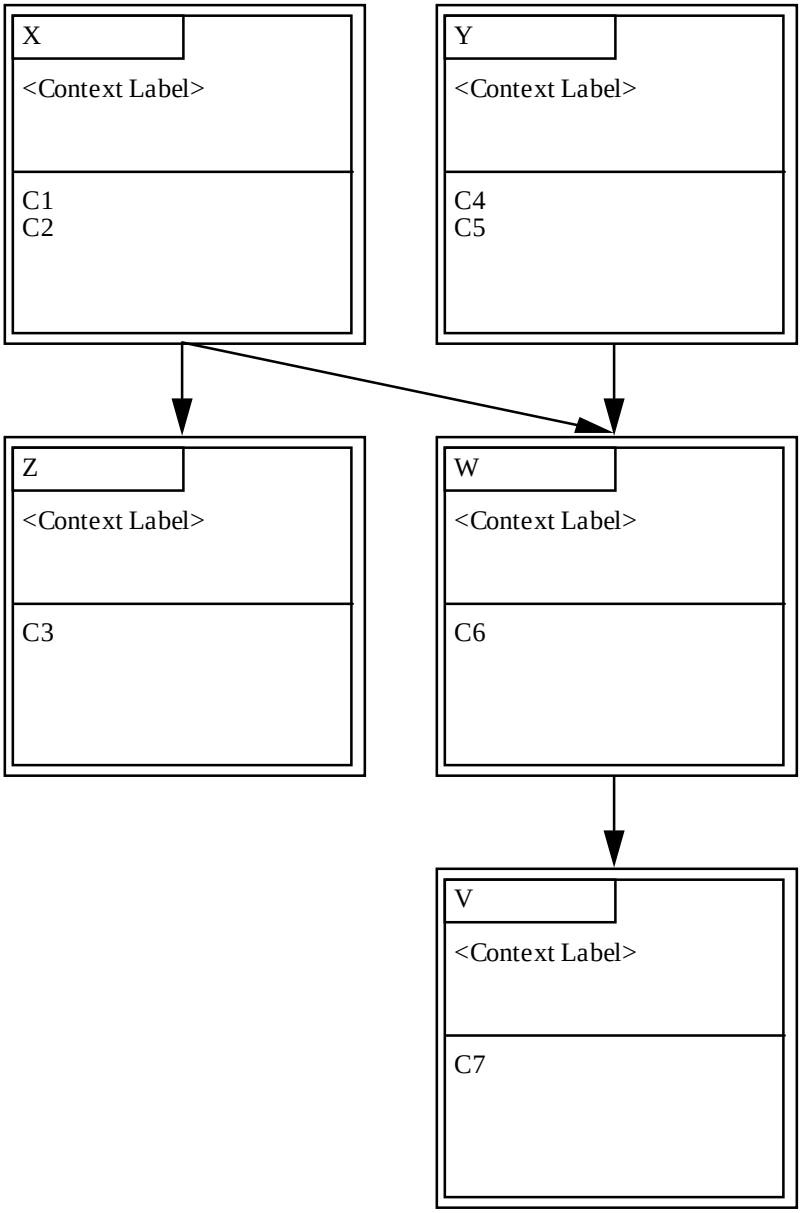


Figure 9.
Example Context Schematic

Candidate Semantics:

- A double-line box denotes a context. The box is divided into top and bottom halves by a separator line with the upper half containing a smaller rectangle anchored to the top left inside corner of the box.

- A single-headed arrow denotes that the context attached to the tail of the arrow subsumes the context attached to the head of the arrow. That is, all constraints that hold in the context attached to the tail of the arrow also hold in the context attached to the head of the arrow.

Discussion:

The example (Figure 9) illustrates five contexts (V, W, X, Y, and Z). Each context is uniquely distinguished by a context identifier and provided with a descriptive label. A more detailed description of the context would be included in an elaboration form supporting the schematic. Although the central focus of this schematic is the context, graphical depiction of the anatomy of the context (i.e., the facts, objects, and relations that collectively define the context) was found to be unnecessary and possibly burdensome. The primary purpose for the schematic is to organize constraints in terms of the situations in which they hold. Thus, the only information necessary to display about the context was that information needed to distinguish one context from another. As indicated by the example above, constraints C1 and C2 are listed in the context box identified as X. The arrow leading from context X to context Y is to show that context Y is a subcontext of context X. That is, all that was true in X is also true in Y plus possibly more constraints. Contexts can be merged to form new contexts as is shown by context W, a combination of context X and context Z. Note that this process is strictly additive. That is, new constraints are always added to the contexts as you move down the schematic. You cannot remove constraints as you move down the schematic.

Issues:

Is there one way of expressing the label for a context that is somehow better than others? Adverbial phrases (e.g., [being] at the construction site, working on government contracts?)

When the list of constraint identifiers grows long, is this an indication that one needs to further refine the context and thus break things up into more manageable pieces?

What pieces of information do we want to record (and/or display) about the contexts and their relationships? What are the types of relationships that we can expect between the concepts? What are the ones that we want to display in the diagram?

Constraint Resource Schematic

Purpose: The purpose of this schematic is to display the systems or objects that maintain or enforce the constraint.

Viewpoint: Constraint-centered

Procedure Components Supported:

- Mode Two, Collect and analyze evidence
- Mode Three, Hypothesize candidate constraints
- Mode Four, Validate and refine constraints

Graphical Metaphor: Hierarchical

Candidate Syntax:

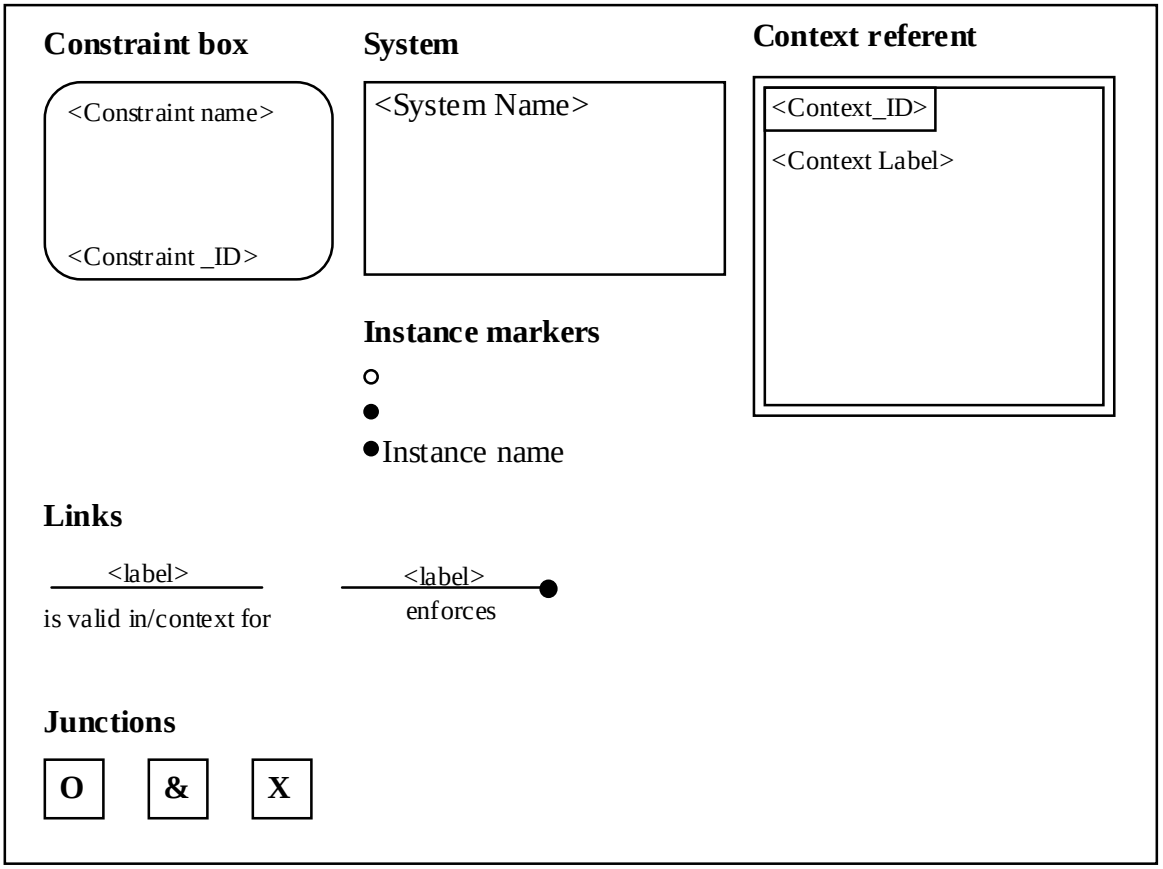


Figure 10.
Candidate Syntax for the Constraint Resource Schematic

Example: Constraint C15: The Board of Directors is responsible for representing the shareholders in ensuring that the allocation of end-of-year profits toward stock value and dividends maintain an acceptable level of Return on Investment (ROI). Acting on a majority vote of the Board of Directors, the Comptroller invests profits and/or distributes dividends to stockholders.

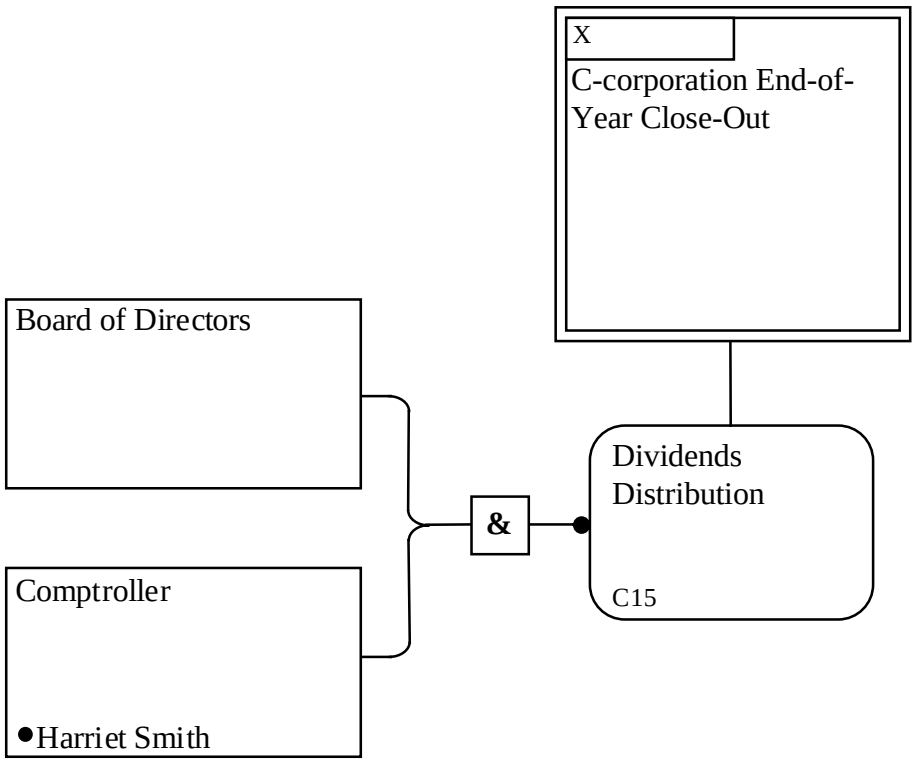


Figure 11.
Example Constraint Resource Schematic

Candidate Semantics:

- A double-lined box denotes a reference to a context.

- A round-cornered box denotes a particular constraint whose unique identifier is located in the lower left-hand corner and whose label is in the upper left hand corner. The label is a meaningful name given to the constraint.

- A rectangle denotes a system (object or collection of objects). Instance markers may be included in the lower left-hand corner of the system rectangle. When no instance marker is present, the system rectangle represents the object kind indicated by the system label. An open instance marker indicates that *all* instances of the object kind are involved. A filled instance marker without a name indicates that *any* one instance of the object kind is involved. A filled instance marker with a name represents the *specific* named instance that is involved.

- As a convenience, the logic symbols could be omitted when the semantics is equivalent to an &-junction.

Note: All links have a label and should have an associated elaboration form that describes, for example, how a constraint is enforced by a system, object, or instance.

Constraint-Relationship Schematic

Purpose: The purpose of this diagram is to display existential dependency, part-of, and user-defined relationships among constraints.

Viewpoint: Relationship-centered

Procedure Component(s) Supported:

- Mode Two, Collect and analyze evidence
- Mode Four, Validate and refine constraints

Graphical Metaphor: Hierarchical

Candidate Syntax:

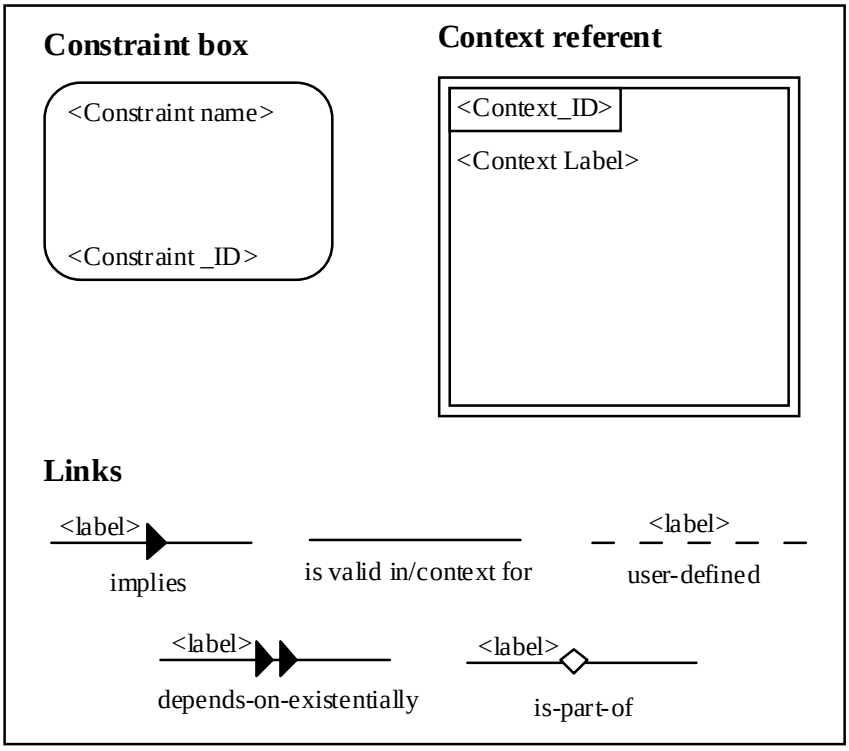


Figure 12.
Candidate Syntax for the Constraint Relationship Schematic

Example:

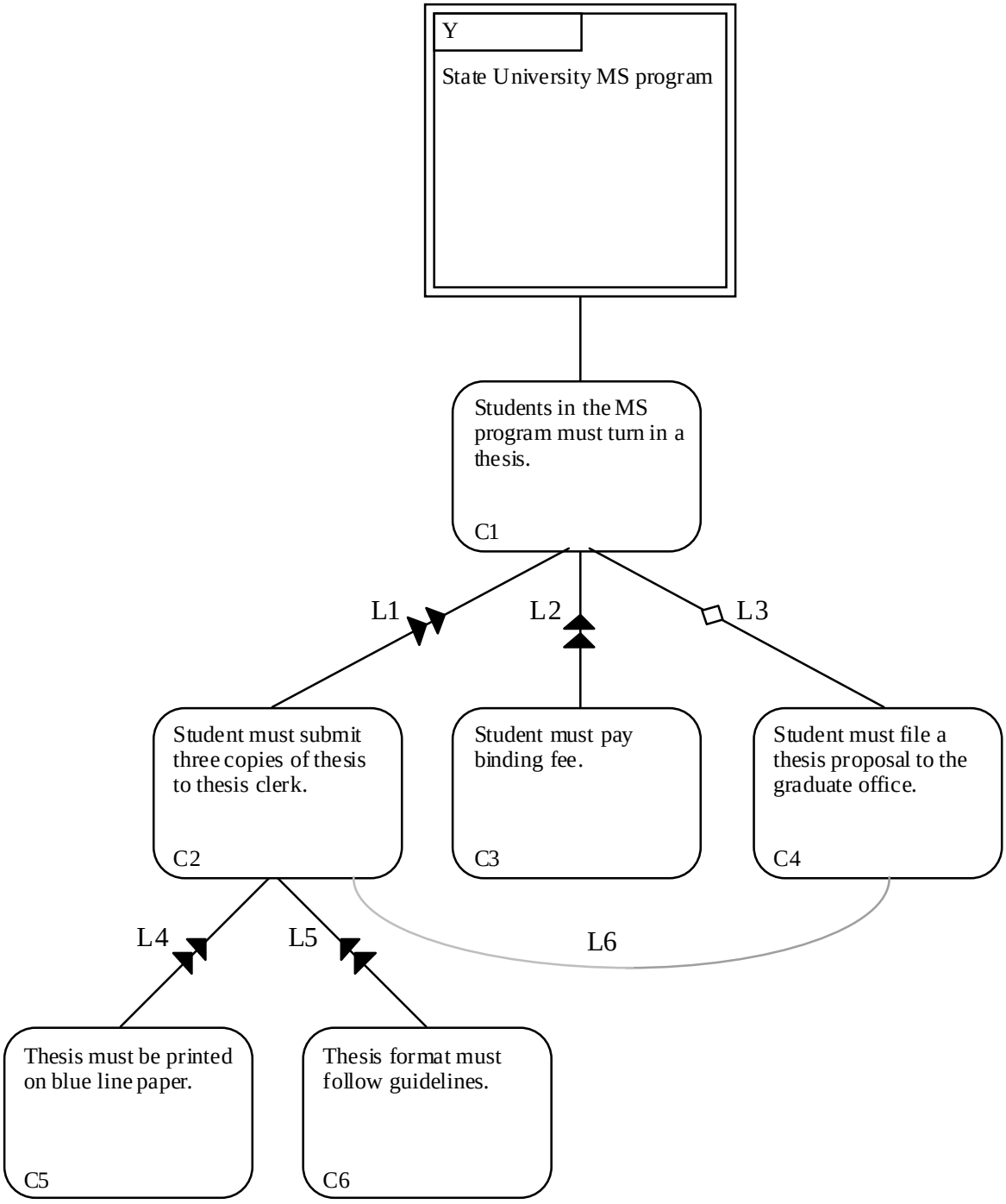


Figure 13.
Example Constraint Relationship Schematic

Candidate semantics:

- A double-lined box denotes a reference to a context.
- A round-cornered box denotes a particular constraint whose unique identifier is located in the lower left-hand corner and whose label is in the upper left hand corner. The label is a meaningful name given to the constraint.

Discussion: The idea for the diagram is to use the high level relation (depends-on) as the main focus. The analyst can specialize these relations using different symbols. Finally, other relations (user-defined) are marked with another symbol.

Issues: All links should have labels and elaboration forms associated with them. The elaboration forms define the nature of the relationship in detail. An important characterization of the relation may be the essential/non-essential property.

Constraint Effects Schematic

Purpose: To display the objects and systems affected by a constraint.

Viewpoint: Constraint-centered

Procedure Component(s) Supported:

- Mode Two, Collect and analyze evidence
- Mode Three, Hypothesize candidate constraints
- Mode Four, Validate and refine constraints

Graphical Metaphor: Spider web

Candidate Syntax:

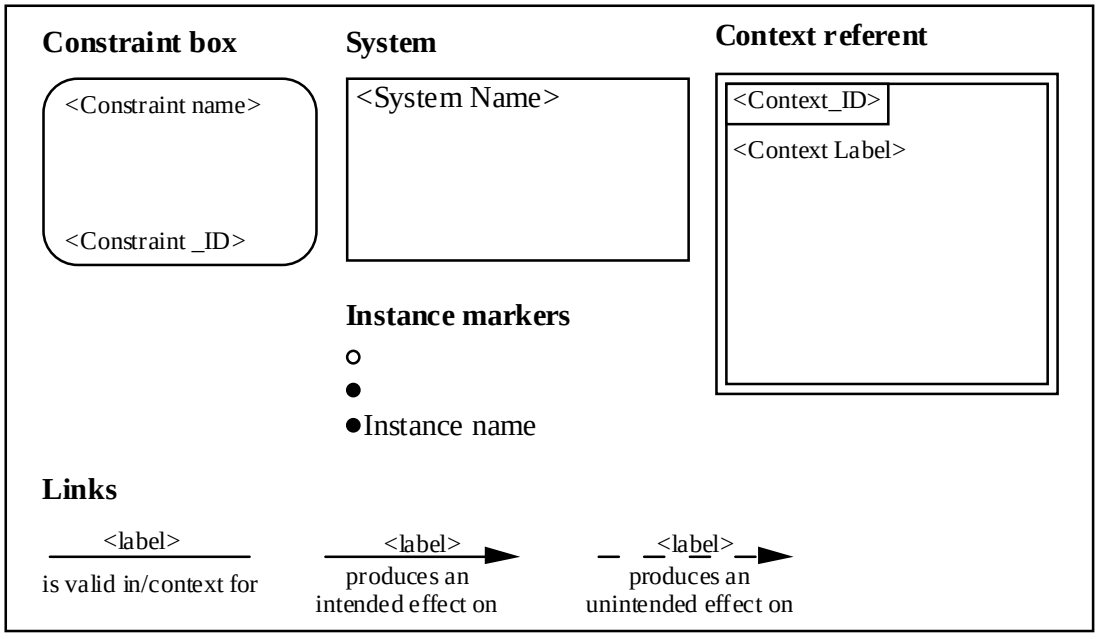


Figure 14.
Candidate Syntax for the Constraint Effects Schematic

Example:

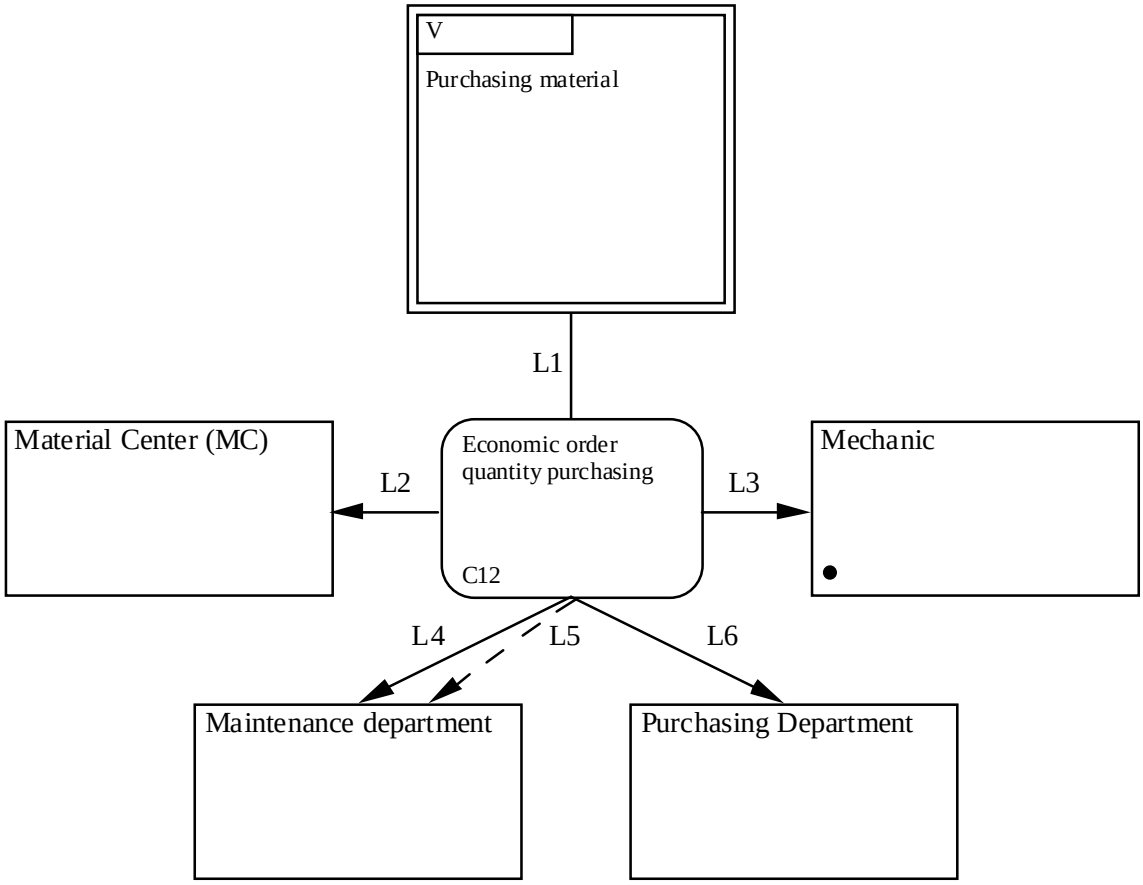


Figure 15.
Example Constraint Effects Schematic

Candidate Semantics:

- A double-lined box denotes a reference to a context.

- A round-cornered box denotes a particular constraint whose unique identifier is located in the lower left-hand corner and whose label is in the upper left hand corner. The label is a meaningful name given to the constraint.

- A rectangle denotes a system (object or collection of objects). Instance markers may be included in the lower left-hand corner of the system rectangle. When no instance marker is present, the system rectangle represents the object kind indicated by the system label. An open instance marker indicates that *all* instances of the object kind are involved. A filled instance marker without a name indicates that *any* one instance of the object kind is involved. A filled instance marker with a name represents the *specific* named instance that is involved.

Goal Schematic

Purpose: The purpose of this diagram is to show the relationship between constraints and individual goals. It may also be used to display the relative impact of each constraint with respect to the goal of interest and in particular the "prioritization" of constraints with respect to the system's goals. The constraints are ordered from top to bottom in "priority" (the highest priority being at the top). The constraints having a negative effect on the

goal are displayed on the left of the goal, while the constraints having a positive effect on the goal are displayed on the right.

Viewpoint: Goal-centered

Procedure Component(s) Supported:

- Mode Two, Collect and analyze evidence
- Mode Three, Hypothesize candidate constraints
- Mode Four, Validate and refine constraints

Graphical Metaphor: Spider/Hierarchical

Candidate Syntax:

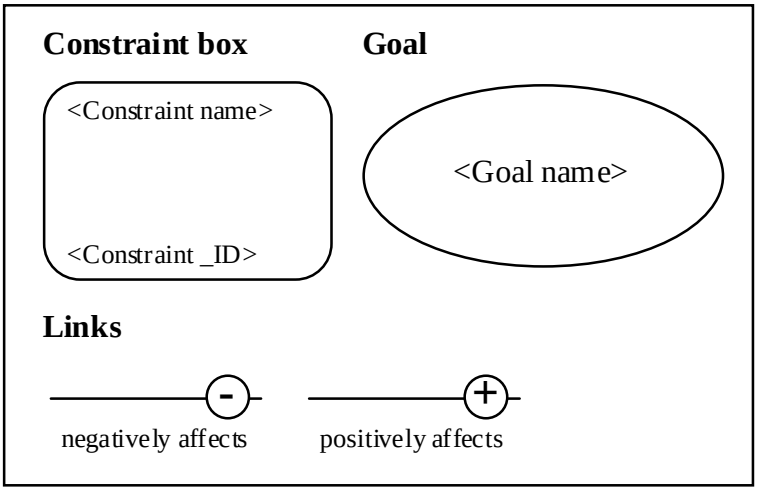
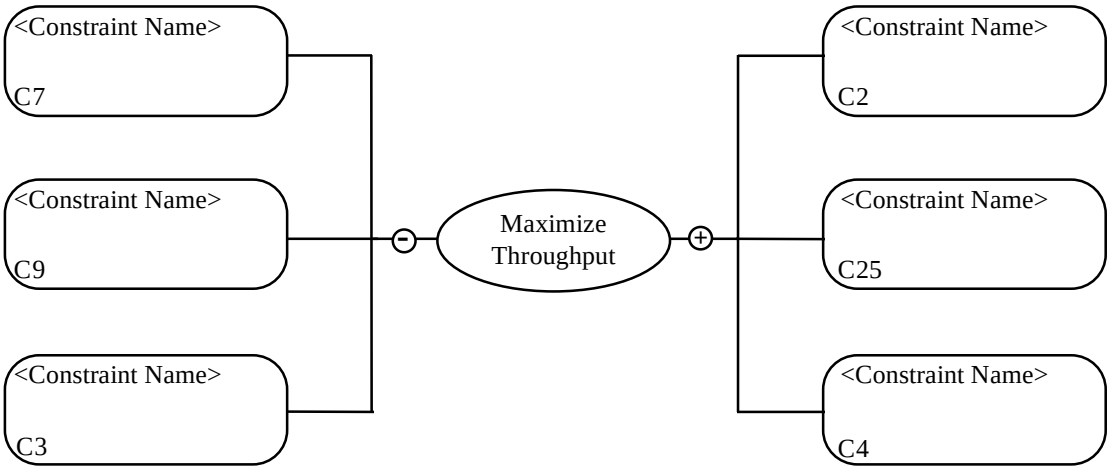


Figure 16.
Candidate Syntax for the Goal Schematic

Example:



or

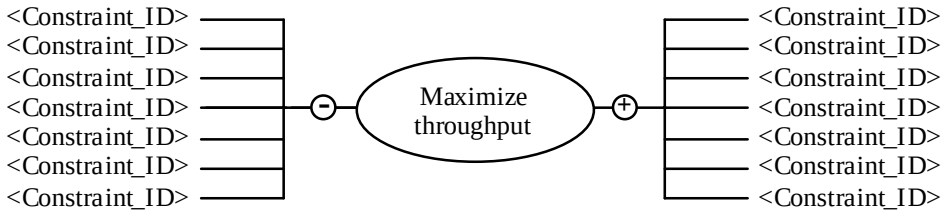


Figure 17.
Example Goal Schematic

Candidate Semantics:

- An oval denotes a goal where the goal is identified by the goal name.

- A rectangle denotes a system (object or collection of objects). Instance markers may be included in the lower left-hand corner of the system rectangle. When no instance marker is present, the system rectangle represents the object kind indicated by the system label. An open instance marker indicates that *all* instances of the object kind are involved. A filled instance marker without a name indicates that *any* one instance of the object kind is involved. A filled instance marker with a name represents the *specific* named instance that is involved.

- For any given pair of constraints on a link, the constraint positioned above the other has a stronger affect.

Discussion: An alternative mechanism for correlating constraints with goals is through the use of a matrix-based metaphor, such as used in Quality Function Deployment diagrams. Both strong and weak positive and negative correlations could be displayed between all goals and constraints, as opposed to centering on a single goal, as with the schematic above.

Goal Relationship Schematic

Purpose: The purpose of this diagram is to show the relationships (subsumption, conflicts, etc.) between goals.

Viewpoint: Goal-centered

Procedure Component(s) Supported:

- Mode Two, Collect and analyze evidence
- Mode Four, Validate and refine constraints

Graphical Metaphor: Hierarchical

Candidate Syntax: To be developed.

Example: None.

Candidate Semantics: Not applicable.

Discussion: A number of relationships between goals can be considered. Among these are *is valid in/context for*, *depends-on-existentially*, *implies*, *is-part-of*, and so forth. The concept of a performance measure may also be needed in the schematic (perhaps as a list of metric names in the goal object).

Issues: To be determined.

Symptom schematic

Purpose: Assist with tracing the underlying cause of symptoms to the lack of an enabling constraint or to the existence of a limiting constraint.

Viewpoint: Symptom-centered

Procedure Component(s) Supported:

- Mode Two, Collect and analyze evidence
- Mode Three, Hypothesize candidate constraints
- Mode Four, Validate and refine constraints

Graphical Metaphor: Spider

Candidate Syntax:

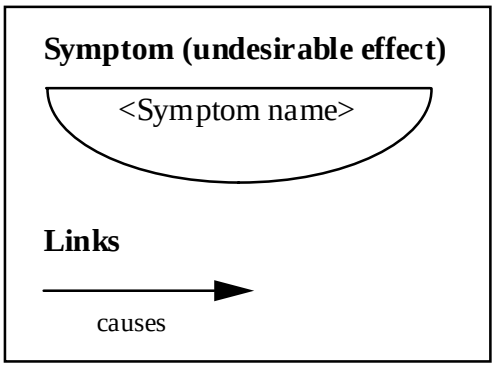


Figure 18.
Candidate Syntax for the Symptom Schematic

Example:

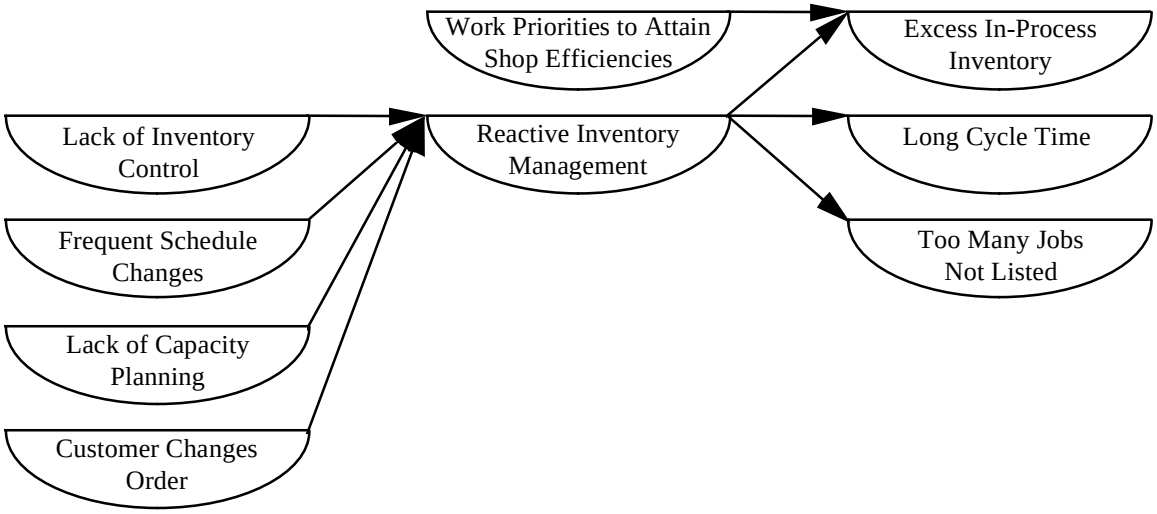


Figure 19.
Example Symptom Schematic

Candidate Semantics:

- A half-oval denotes a symptom or negative effect that is distinguished by its label.

- An "inclusive or" condition is implied at any point where arrow heads or tails converge. In other words, either one, a combination, or all of the symptoms stand in the relation "causes/is caused by" when multiple arrows originate from, or terminate at, a symptom symbol.

Discussion: A symptom is defined as an indication or sign (e.g., change in normal function, sensation, or appearance) indicating a problem. A problem is defined as a source of trouble or annoyance. To one domain expert, one problem is the cause of another problem; while to another individual the same problem will be recognized as merely a symptom of another problem. Thus, the way negative effects are described (i.e., as the cause, problem, or effect) depends on individual viewpoints. The actual root cause(s) of negative effects (or positive effects) can be very elusive. Symptoms and causality relationships among symptoms, on the other hand, are relatively easy for domain experts to recognize. For this reason, the schematic above has been named the *symptom schematic*.

Issues:

- Given an *inclusive-or* semantics on the relation *causes/is caused by* in the above schematic, additional information will need to be captured on a supporting elaboration form. This would help make the relations explicit, provided that information was available. An alternative might be to explore the introduction of new graphical symbols to display additional logic on the schematic.

- It may be useful to provide a mechanism to classify sets of like symptoms.

- An alternative schematic that merits further investigation is the Ishikawa diagram, also called a fishbone or cause-effect diagram, to assist in discovering the relationship between undesirable effects. Ishikawa diagrams are single problem-centered (i.e., one problem, symptom, or effect is isolated in the diagram as the center of focus). The central problem displayed on Ishikawa diagrams is represented by a "backbone" arrow from which one or more branches and sub-branches are created to illustrate cause-effect chains terminating at the backbone. The schematic above allows for either single effect-centered analysis and data collection or multi-effect analysis.

Significant Accomplishments

Among the most significant contributions of the Armstrong Laboratory's IICE program is a characterization of the nature and role of constraints in governing and predicting behavior among objects and agents in a system. This contribution is underscored by the observation that many domain experts need to "unlearn" an association between *constraint* and something negative. The majority of constraints, in fact, are necessary and enabling. The IDEF9 method helps to promote awareness of both the enabling and limiting aspects of constraints in an organizational context.

Building upon this foundation, the need for a method promoting a systematic approach to business constraint discovery was established. The work that followed has produced a prototype procedure and graphical language uniquely suited to the task. Together, these components establish an explicit process for recognizing, collecting, documenting, and validating business constraints.

Among the central features of the procedure is the provision for identifying *type problems* typical of business systems (See Table 1). The procedure itself was also designed to help avoid analogous problems in IDEF9 method application. A validation procedure was developed wherein analysts first hypothesize candidate constraints and then attempt to substantiate them with supporting data. This is followed by domain experts challenging the conclusions of the analyst. The challenge step helps to ensure that candidate constraints promoted to the status of constraint become so through a participative and consensus-building activity with domain experts.

Explorations into alternative language designs uncovered a number of promising schematics, each with a unique focus supporting constraint discovery and downstream reuse of constraint information. One of the most interesting of these is the Context Schematic. The simplicity of this schematic in representing the context in which a constraint holds illustrates the point that a language is often best judged not by what can be expressed but by what is explicitly not expressed. The structure of a context in terms of the objects, facts, and relations comprising the context could have easily overwhelmed the design of this schematic. The context schematic permits successive levels of detailing that allow one to describe the context at an appropriate level of granularity. This design also permits focus on the relationship between the constraints and the context(s) in which it holds rather than placing an inordinate emphasis on the context itself.

The Constraint Effects Schematic illustrates the relationship between constraints and affected objects in terms of intended and unintended effects. Existential dependency relations among constraints are displayed in the Constraint Relationship Schematic permitting rapid identification of outdated or unnecessary constraints. The Goal Schematic provides a mechanism to establish the relationship between constraints and organization goals and for prioritizing their relative impact. The Symptom Schematic assists in identifying cause-effect chains among problematic symptoms that can ultimately lead to the discovery of constraints.

Collectively, these developments provide a foundation for future development. Additional development, testing, and refinement will be needed, however. The following section outlines some potential areas for future development, both within the IDEF9 method and in terms of spin-off work that can provide additional capabilities to leverage constraint information.

Potential Areas for Future Work

A number of promising areas for additional work were identified during the development of the IDEF9 method. Several of these are listed below together with a brief description of the benefits to be gained by pursuing further development along these lines.

1. Method refinement. A number of areas within the prototype method merit further development and testing. Among these are expansions to the techniques supporting the IDEF9 method procedure, further development of graphical language elements, the incorporation of elaboration forms and data collection sheets, and the development of a computational language of expression (elaboration language). Specific provisions were made to ensure ease of integration with other methods (particularly with the IDEF5 method); however, a more in-depth treatment of this aspect of IDEF9

merits consideration. Each of IDEF9's design elements also needs to be tested, in a highly controlled setting and in a production environment where unanticipated misuse or other problems can be detected and resolved. A wide range of test situations is recommended to maximize the robustness of the method.

2. Application framework development. Methods are, by design, highly task or skill oriented. Their true value is demonstrated mainly through their application in projects involving many skills and tasks coordinated to satisfy some goal or objective. Constraint discovery is an implicit first step for strategic planning, BPR, TQM, and a number of similar improvement strategies. There is a need, however, to establish exactly how constraint discovery activity and the constraint information collected can be integrated into the process.

3. Argument validity checking. Most decision-making is based on qualitative judgments, not on quantitative data (e.g., bottom-line cost). Arguments for or against a proposed decision establish a case for action, one of whose supporting elements may be a business case. The relative merits of one decision over another is largely determined by the validity of the argument posed to support the decision. Arguments are themselves composed of a chain of premises and conclusions that eventually lead to the final conclusion. Further investigation could be made to provide the IDEF9 method with techniques specifically enabling decision-makers to assess the validity of a proposed argument.

4. Mechanisms to allocate costs for enforcing constraints. Activity Based Costing (ABC) has been successfully applied as a mechanism for allocating costs to organization functions and for determining the appropriate cost of doing business [Kaplan 88]. The basic mechanism underlying the ABC approach is to identify how resources are used by activities to accumulate costs. The activity costs are then traced to the products and services generated by the activities. It would be useful to explore using a similar approach to allocate costs to constraints since constraints use resources in their maintenance or enforcement. Such mechanisms would provide additional decision support for determining when the cost of a constraint exceeds its value and when it merits additional emphasis.

5. Methods and tools to design constraints. The design or redesign of constraints is a potential research topic in which a number of promising areas can be explored. The purpose of such efforts would be to assist with change management through more effective, proactive constraint management. Methods and tools are needed to support the design of constraints such that intended effects are maximized, unintended effects are minimized, systems designed to enforce constraints are appropriately configured, constraints are "flagged" whenever changes in the environment precipitate the need to reevaluate the need for a constraint, and so on.

6. Constraint categorization schemes. The mental exercise of classifying constraints has been shown to assist in constraint discovery and downstream reuse of constraint information. Further research is needed, however, to explore alternative constraint taxonomies and their use in analyzing candidate constraints, checking for appropriate coverage in constraint discovery effort, and in identifying opportunities for improvement. The product of such effort would be a set of constraint taxonomies and techniques for using the taxonomies to support varied analyses.

7. Tools to capture, display, and maintain constraint information. The success of any method depends heavily on the existence of automated tools. This has always been true and is likely to continue in the foreseeable future. Automated tools not only assist practitioners in the application of a method but provide a rapid and reliable means for sharing, storing, and reusing information.

8. Libraries of constraints. On-line repositories of business constraints made available through the information superhighway would provide business owners, strategic and tactical planners, and systems developers with a mechanism for exploring new ways of doing business and for expanding into new areas of business. The goals of dual-use conversion, agility, and similar initiatives depend on the ability to expose the current constraints under which the business operates and the constraints under which world-class systems in a given industry sector operate. With this visibility, a clear path to leveraging new areas of opportunity can be established.

9. Tools and environments for constraint-driven change management. One of the key problems facing organizations today is not having the ability to understand the effect of local change on the global enterprise. Many times a small change in policy, procedure, or product in one area has adverse effects in other areas. Decision-makers must then reverse the initial change or live with the ripple effects. Either of these situations results in an inefficient use of resources. In organizations today, significant effort is expended on managing the effects of change, not the management of change itself. To reverse this situation, there is a need for more visibility of constraining relationships giving rise to organization behaviors, predictive tools enabling assessment of the impacts of change, and design tools assisting in the development of the system of constraints needed to proactively manage change.

Conclusions

Effective change management is greatly facilitated through the discovery and documentation of business constraints. The change management process begins with identifying business constraints. The knowledge of what constraints exists and how those constraints interact is generally incomplete. Yet business constraints initiate, enable, govern, and limit the behavior of objects and agents to accomplish the goals of the business. The IDEF9 method was designed to assist in the discovery and analysis of these constraints.

Considerable progress has been made toward the development of IDEF9. In its current form, IDEF9 provides a systematic approach for business owners, strategic and tactical planners, systems developers, project leaders, and decision-makers to identify and document business constraints. These developments provide the foundation for future endeavors to refine the IDEF9 method and to leverage the products of IDEF9 application.

IDEF9 Bibliography

[Demmy & Petrini] Demmy, S. W., & Petrini, A. B. (1993). The Theory of Constraints: A New Weapon for Depot Maintenance Planning and Control. *Air Force Journal of Logistics*.

[Goldratt & Cox 86] Goldratt, E. M., & Cox, J. (1986). *The Goal*. Croton-On-Hudson, NY: North River Press.

[Goldratt & Fox 86] Goldratt, E. M., & Fox, R. E. (1986). *The Race*. Croton-On-Hudson, NY: North River Press.

[Goldratt 90] Goldratt, E. M. (1990). *Theory of Constraints*. Croton-On-Hudson, NY: North River Press.

[Goldratt 90] Goldratt, E. M. (1990). *The Haystack Syndrome*. Croton-On-Hudson, NY: North River Press.

[Goldratt 92a] Goldratt, E. M. (1992, July). *An Introduction to Theory of Constraints: The Production Approach--Workshop Description*.

[Goldratt 92b] Goldratt, E. M. (1992, July). *An Introduction to Theory of Constraints: The Production Approach--Two-Day Workshop Course Materials*.

[Coleman 91] Coleman, S. (1991). *Information Engineering Practitioner's Guide Volume IV Business Area Analysis* Culver City, CA: Pacific Information Management, Inc.

[Coleman 92] Coleman, S. (1992). *Corporate Information Management Process Improvement Methodology for DoD Functional Managers*, Unpublished report by D. Appleton Company, Inc.

[Gross 87] Gross, M., Ervin, S., Anderson, J., & Fleisher, A. (1987) Designing with constraints. In Y.E. Kalay (Ed.), *Computability of Design*. New York: Wiley.

[Martin 82] Martin, J. (1982). *An Information Systems Manifesto.* Englewood Cliffs, NJ: Prentice-Hall, Inc.

[Maher 89] Maher, M. L. (1989). Synthesis and evaluation of preliminary designs. In J. S. Gero (Ed.), *Artificial Intelligence in Design*. New York: Springer-Verlag.

[Mayer, R. J., et al. 87] Mayer, R. J., et al. (1987). *Knowledge-based integrated information systems development methodologies plan* (Vol. 2), (DTIC-A195851).

[Olle, T. W., et al. 88] Olle, T. W., et al. (1988). *Information Systems Methodologies: A Framework for Understanding*. Reading, MA: Addison-Wesley Publishing Company.

[Pressman 87] Pressman, R. S. (1987). *Software Engineering: A Practitioner's Approach*. New York: McGraw-Hill, 1987.

[Ramey 83] Ramey, T. L. (1983). *Guidebook to System Development*. Hughes Aircraft Company.

[Silver & Silver 89] Silver, G. A., & Silver, M. L. (1989). *Systems Analysis and Design*. Reading, MA: Addison-Wesley.

[Smith & Browne 93] Smith, G. F., & Browne, G. J. (1993). Conceptual Foundations of Design Problem Solving. *IEEE Transactions on Systems, Man, and Cybernatics*, 23(5).

[Smith 93] Smith, G. F. (1993). Defining Real World Problems: A Conceptual Language. *IEEE Transactions on Systems, Man, and Cybernatics*, 23(5).

[Wallace 87] Wallace, R. H., Stockenberg, J. E., & Charette, R. N. (1987). *A Unified Methodology for Developing Systems*. New York: McGraw-Hill, Inc.

[Zachman 87] Zachman, J. (1987). A framework for information systems architecture. *IBM Systems Journal*, 26(3), 276-292.

Glossary of IDEF9 Terms

Constraint	A relationship that is maintained or enforced in a given context.
Context	A distinguished set of conditions.
Project scope	A delineated set of boundaries for constraint discovery effort documented in the form of scope statements.
Project purpose	The aim, object, or desired ends for the constraint discovery effort. The purpose is usually established by prioritized objective statements for the effort, statements of needs that the constraint discovery effort must satisfy, and questions or findings the client wants answered.
Evidence	An indication, sign, or manifestation that supports the existence of a constraint in a given context.
Relation, relationship	An abstract, general association or connection that holds between two or more conceptual or physical objects.
Effect(s) of a constraint	Something that inevitably follows an antecedent (as a cause or agent). Effects are classified as direct or indirect, intended or unintended, and desirable or undesirable.
Symptom	Something characteristic or indicative of a condition impairing normal functioning.
Process	An ordered sequence of events. In human-designed systems, the events that constitute a process are designed and ordered to achieve some desired outcome. A business process, in particular, is an ordered sequence of events involving people, materials, energy, and equipment, that is designed to achieve a defined business outcome.
System	A collection of objects standing in particular relations and exhibiting particular behavior prescribed by a collection of constraints.
System, business	A collection of objects behaving to perform one or more business functions under the influence of constraints to accomplish a particular goal.
Goal	An object or end that the business or system strives to achieve.
Constraint, rationale of the	The set of beliefs motivating the establishment and maintenance of a constraining relationship.
Constraint statement	A textual description of the constraining relationship.

TOWARD A DESIGN RATIONALE CAPTURE METHOD (IDEF6)

Introduction

The purpose of the IDEF6 Design Rationale Capture method is to facilitate the acquisition, representation, and manipulation of the design rationale used in the development of enterprise systems. Rationale is the reason, justification, underlying motivation, or excuse that moved the designer to select a particular strategy or design feature. More simply, rationale is interpreted as the answer to the question, "Why is this design being done in this manner?" Most design methods focus on the what the design is (i.e., on the final product, rather than why the design is the way it is).

Design rationale becomes important when a design decision is not completely determined by the constraints of the situation. Thus, decision points must be identified, the situations and constraints associated with those decision points must be defined, and if options exist, the rationale for the chosen option and for discarding other options (i.e., those design options not chosen) must be recorded.

The task of capturing design rationale serves the following purposes:

1. Enables evolutionary integration of enterprise information systems.

2. Enables the use of concurrent engineering methods in information systems development.

3. Supports better integration among life-cycle artifacts.

4. Facilitates business reengineering by capturing the rationale behind business case decisions.

5. Enables efficient traceability of decisions.

Rationale capture is applicable to all phases of the system development process. The intended users of IDEF6 include business system engineers, information systems designers, software designers, systems development project managers, and programmers.

Motivation

Supporting the evolution of integrated information systems whose conceptualization, construction, operation, maintenance, and retirement may span careers, or even lifetimes, requires the explicit capture and storage of design rationale. Personnel turnovers and manpower shifts create similar needs for design rationale capture. In addition, capturing design rationale is important during the development phase of large scale systems. In these situations, the logic (i.e., the chain of arguments or reasons) behind the design is invaluable to the down-stream developers, testers, and integrators.

Computer-aided software engineering (CASE) environments attempt to bring automated support to the design stage. Existing CASE tools are inherently limited in at least two important respects. CASE tools are intended to document various aspects of *what* a design is, but they do not document *why* a design exists. Secondly, even when design rationale comments exist, they are captured as unstructured textual comments. Often, the only record of the design rationale for a software system is distributed across many people, and at any one time, parts of it will have been forgotten or made unavailable.

The loss of design rationale can result in repeating past mistakes or making decisions contrary to earlier design decisions. Furthermore, when one does not understand the rationale for why the system is the way it is, actions taken to correct one problem may cause more problems elsewhere in the system.

One benefit of maintaining design rationale is to force a statement of goals, as well as assumptions. Goals, like assumptions, are frequently not stated. Forcing their statement for the purpose of rationale capture leads to

a more focus2ed, disciplined approach to design. Much of design thinking appears to be abductive in nature, with experience-directed insights being fashioned and rationalized in the context of the current task. This rationalization may be the only basis for understanding why a system is the way it is. Without the capture of this chain of reasoning, communication of the design becomes difficult and prone to error.

Another motivation for rationale capture is that the definition of subsystems and subsystem boundaries is an experimentation process in which each designer discovers the boundaries he or she finally imposes. If the organization is to avoid costly errors, it must have knowledge of the paths of inquiry that failed, the path to the final success, and the final result.

Benefits

Potential applications of IDEF6 Design Rationale Capture include software design support, software implementation and maintenance support, simulation design, decision support, diagnostics support, and reverse engineering environments.

The potential benefits of a method for design rationale capture include:

1. New data for information system design theory development.
2. Improved reusability of information systems models and designs.
3. Reusability of rationale.
4. Easier system maintenance.
5. Improved collaboration support.
6. More effective decision making.
7. Improved support for evolutionary systems integration.

Summary of Developments and Research Findings

This section describes the nature of design rationale and associated issues that were investigated in the development of IDEF6. The most important factor in the development of IDEF6 was to make the rationale capture as easy and unobtrusive as possible so that it will not disrupt the natural flow of the design process. We also discuss procedure and language design developments.

The basic strategy used to explore IDEF6 method design involved two activities. The first focused on defining the characteristics and the nature of design rationale by analytic means. This involved characterizing the types of information that are used to rationalize a design. This required building an ontology of design rationale. The second activity focused on defining the characteristics and nature of design rationale through direct experimentation. This activity also involved experimenting with language structures designed around the concepts identified in the first activity. Specifically, we experimented with an initial IDEF6 annotation format for the IDEF4 Object-Oriented Design Method. IDEF4 models were extended to include IDEF6 annotations describing assumptions, constraints, or usage categories underlying IDEF4 entities. These investigations were expanded to explore explicit language support for different kinds of justifications, different kinds of constraints, and for individual design artifacts produced using other IDEF methods.

IDEF6 Basic Concepts

Design rationale (why and how), can be contrasted with the related notions of design specification (what), and design history (steps taken). Design specifications describe what intent should be realized in the final physical artifact. Design rationale describes why the design specification is the way it is. This includes such

information as principles and philosophy of operation, models of correct behavior, and models of how the artifact behaves as it fails. The design process history records the steps that were taken, the plans and expectations that led up to these steps, and the results of each step.

Design Rationale Phenomena

A general characterization of design rationale can be given as: "The beliefs and facts as well as their organization that the human uses to make (or justify) design commitments and to propagate those commitments."

In our investigation into the nature of design rationale, we have characterized "types" of design rationale and "mechanisms" for representation of these types. Types of design rationale include arguments based on:

(1) The philosophy of a design, including:

  (a) Process descriptions of intended system operation, and

  (b) Design themes expressed in terms of particular object types standing in some specific relation or exhibiting some specific behavior.

(2) Design limitations expressed as range restrictions on system parameters, and environmental factors.

  (3) Factors considered in tradeoff decisions, including budget constraints, timing constraints, organization policies and procedures, and available technology.

(4) Design goals:

  (a) Use or lack of use of particular components,

  (b) Achievement of particular structural arrangements,

  (c) Priorities on problems requirements,

  (d) Product life cycle characteristics (e.g., disposable versus maintainable, robustness, flexibility), and

  (e) Design rules followed in problem or solution space partitioning, test/model data interpretation, or system structuring.

(5) Precedence or historical proof of viability.

(6) Legislative, social, professional society, business, or personal evaluation factors or constraints.

Possibly due to the commonness of routine design or the complexity of design rationale expression, the most common rationale given for a design is that a similar design worked in a previous situation. A minimum requirement for a design knowledge management capability is that it must be able to record historical precedence, as well as statements of beliefs and rationalizations for why a current design situation is identical to a previous one.. This phenomena may be less common in software design rationale. The malleability of design gives rise to the belief that designers can be creative each time. In contrast with hardware design, software design based on previous designs means that reused parts are literally copied whole, not rebuilt from the same plans. Hence, the opportunity to fix small design flaws is lost, and the interaction of the new parts with the old is likely to be much less well understood. Another important rationale given for a design is that "it feels better," or "it seems more balanced, symmetric." In other words, there is an important aesthetic side to software design.

Finally, software design rationale includes expectations about how the design will evolve through the development process. For example, there may be expectations about how the program structure will change. These expectations do not appear to be as well defined as similar expectations we have seen in mechanical hardware design.

One general conclusion made about the nature of design rationale is that it traces the reasoning process. This trace starts with the element of the design that is being justified and provides a set of supporting arguments that ultimately terminate with proven elements of the problem space or the design space. This chain of arguments may also terminate in previously rationalized design elements.

Design Rationale Capture Issues

One reason for the loss of rationale is rooted in the long time lag between specification of the software artifact and completion of the artifact.

There are also problems with developing a general understanding of what should constitute explicitly captured Design Rationale. That is, a notable difficulty with expressing design rationale is that the concept itself is not uniformly understood. It shares this characteristic with all other forms of "explanation" that Artificial Intelligence (AI) researchers continue to struggle with.

One of the problems with capturing design rationale is that it requires stating characteristics beyond the minimum specifications required to produce the product. The major goal of design has traditionally been to construct specifications for artifacts so complete that any realization of them will satisfy the requirements (and thereby solve the problems). The logic behind the decisions leading to, resulting in, or contributing to the design are not the main focus of design and are generally left unrecorded. After all, it has been thought that their inclusion into the traditional document structures may cause confusion or complicate the acquisition/interpretation of critical information communicated through the documents. In addition, , a designer may make hundreds of focused component decisions and thousands of configuration decisions in a very short period of time [Friel 88, Goel 90]. Lack of efficient methods to capture and represent these decision alternatives and considerations are a primary impediment to the design rationale capture.

IDEF6 Procedure Developments

In IDEF6, the rationale capture procedure involves partitioning, classification/ specification, assembly, simulation/execution, and re-partitioning activities. The rationale capture procedure normally applied in the simulation/execution activity of the evolving design uses two phases: Phase I describes the problem and Phase II develops a solution strategy . Initial IDEF6 developments involved experimenting with an IDEF6 annotation format for the IDEF4 Object-Oriented Design Method. IDEF4 models were thus extended to include IDEF6 annotations describing assumptions, constraints, or usage categories underlying IDEF4 entities. We therefore use IDEF4's design rationale component in this report to represent the state of IDEF6 development.

Design is an iterative procedure involving partitioning, classification/specification, assembly, simulation, and re-partitioning activities (see Figure 21). First, the design is partitioned into design artifacts. Each artifact is either classified against existing design artifacts or an external specification is developed for it. The external specification enables the internal specification of the design artifact to be delegated and performed concurrently. After classification/specification, the interfaces between the design artifacts are specified in the assembly activity (i.e., static, dynamic, and behavioral models detailing different aspects of the interaction between design artifacts are developed). While the models are developed, it is important to simulate use scenarios or cases [Jacobsen 94] between design artifacts to uncover design flaws. By analyzing these flaws, the designer can re-arrange the existing models and simulate them until the designer is satisfied. The observed design flaws and the actions contemplated and taken for each are the basis of the design rationale capture procedure.

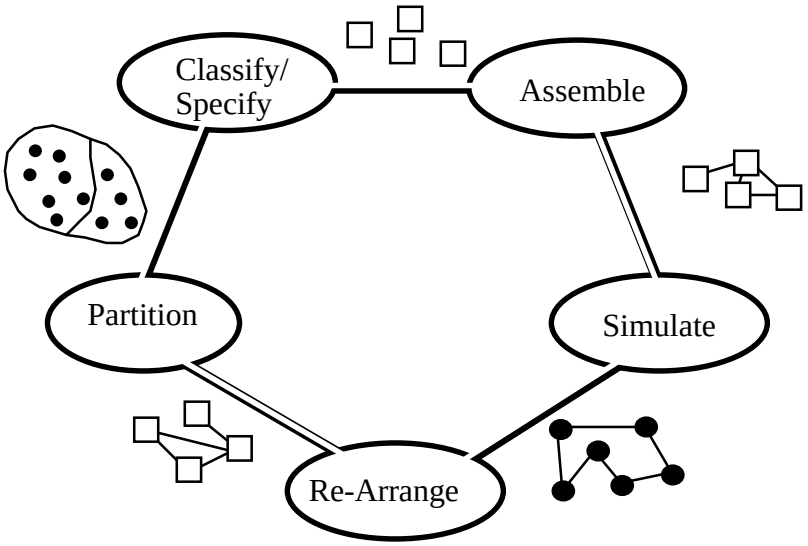


Figure 20.
IDEF4 Design Activities

Phase I: Describe Problem

Problems with the current state of the design are normally identified during the simulation activity of the design cycle. In the simulation activity, use scenarios are used to validate and verify the design or implementation. In Phase I, the designer describes problems that exist in the current design state by: identifying problems, identifying violated constraints (requirements, physical laws, norms, etc.), identifying needs for problem solution, and formulating goals and requirements for the subsequent design iteration.

Identify Problems

The designer identifies problems in the current design state by stepping through the use cases in the requirements model to validate that the design satisfies requirements and to verify that the design will function as intended. The designer records symptoms or concerns about the current design state. A symptom is an observation of an operational failure or undesirable condition in the existing design. A concern is an observation of an anticipated failure or undesirable condition in the existing design.

The requirements model consists of a function[12] model and several process[13] models that describe intended system usage. The function model constrains both the scope of the design partition and the activities supported by the partition. The process models describe use scenarios of how the activities occur. The activities in the function models may call out[14] process models. These models map to requirements and goals. The function use model is used for validating and verifying interfaces and activities, and the process use model is used for validating and verifying process flow and logic.

Figure 21 depicts a design for a design partition called Sys, showing its constituent static and dynamic models, as well as its associated requirements model. The requirements model contains an IDEFØ function model whose activities call out IDEF3 process scenarios. The designer walks the design through these use cases to detect situations that have not been adequately supported.

---

[12] Function models may be expressed in IDEFØ, but should not contain decompositions by type (of activity).

[13] Process models may be expressed in IDEF3.

[14] IDEFØ activities call IDEF3 process scenarios using call mechanism arrows.

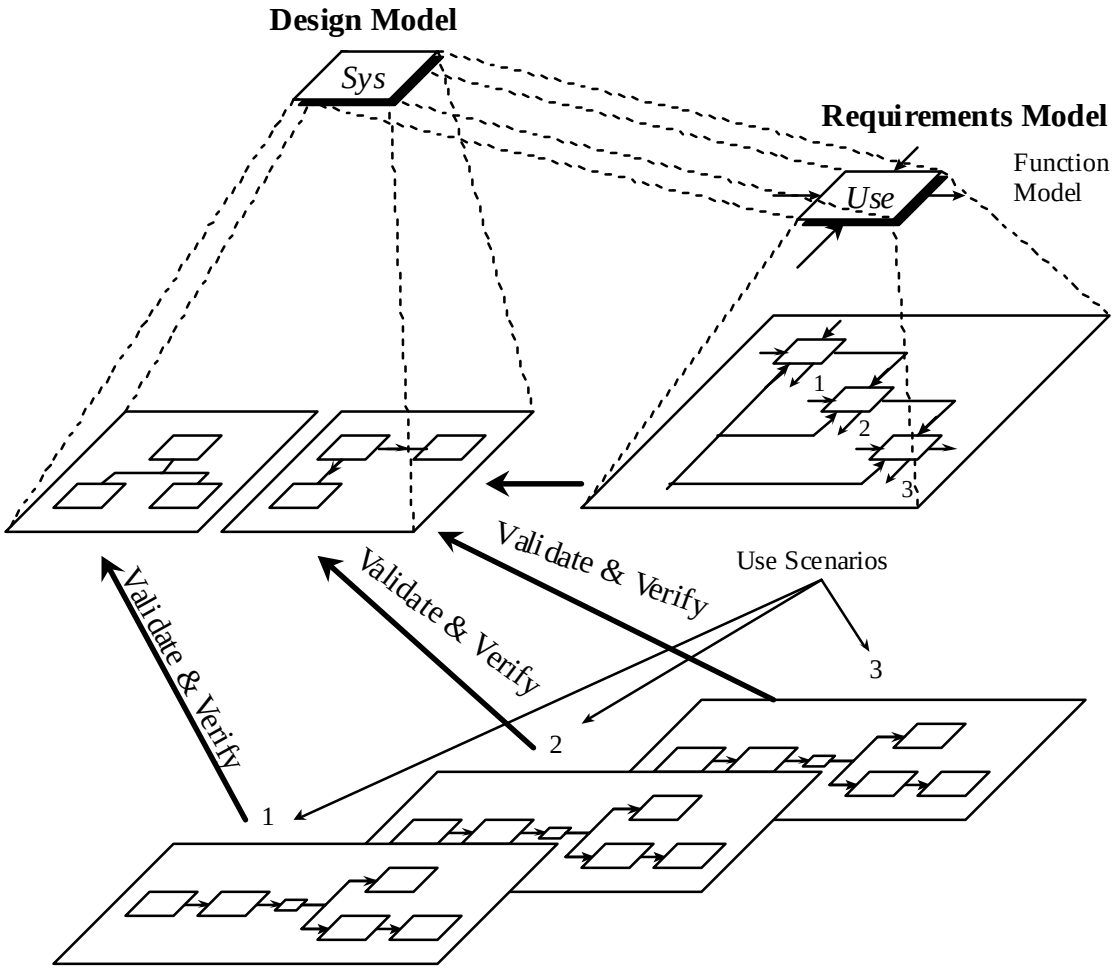


Figure 21.
Static, Dynamic, and Requirements Models for Sys Partition

Identify Constraints

The designer then identifies the constraints that the problems violate or potentially violate. These constraints include requirements, goals, physical laws, conventions, assumptions, models, and resources. Because the activities and processes in the use case scenarios map to requirements and goals, the failure of the design in any use case activity or process can be traced directly to requirements statements and goal statements.

Figure 22 illustrates the mapping between the analysis model's function and use scenarios and the requirements and goal statements. When the design fails to adequately support activities and use scenarios, the requirements model allows the designer to easily identify the requirements constraints or goal statements that have been violated.

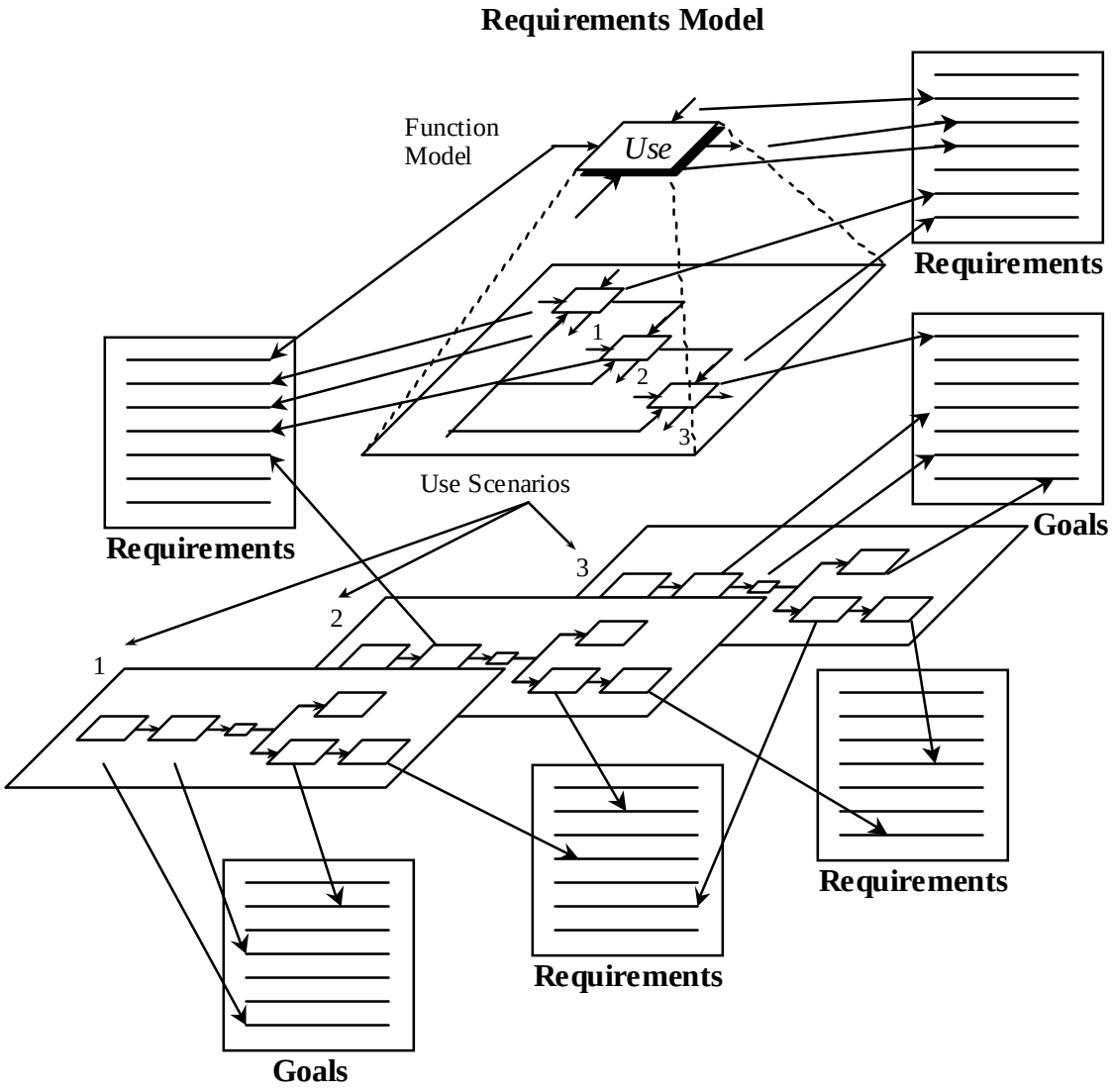


Figure 22.
Functions and Use Scenarios Mapping to Requirements and Goals

Identify Needs

The designer then identifies the necessary conditions or needs for solving the problems. A need is a necessary condition that must be met if a particular problem or set of problems is to be solved. It is possible that the needs statement will have to describe the essentiality for relaxing requirements and goal constraints governing the design.

Formulate Goals and Requirements

Once the needs for the design transition have been identified, the designer formulates (1) requirements that the solution must satisfy and (2) goals that the solution should attempt to satisfy. A requirement is a constraint on either the functional, behavioral, physical, or method of development aspects of a solution. A design goal is a stated aim that the design structure and specifications must support.

Phase II: Formulate Solution Strategies

Once the requirements and goals have been established, the design team formulates alternative strategies for exploration in the next major transition in the design.

One important aspect that distinguishes good designers is the ability to choose between making strategic design decisions and tactical design decisions. Strategic design decisions have sweeping architectural implications (i.e., the decision to use an OODBMS, the separation of responsibilities in client server architecture, the choice of mechanism for inter-process communication). Tactical decisions represent essential details that complete architectural decisions (i.e., the schema of a database, the protocol of a class, and the signature of a member function).

Design strategies can be considered as "meta-plans" for dealing with frequently occurring design situations. They can be viewed as methodizations or organizations of the primitive design activities identified above (i.e., partitioning, classification/specification, assembly, simulation, and re-partitioning). The three types of design strategies considered in the IDEF4 rationale component include:

1. **External-constraint-driven design**—Design carried out under situations where the goals, intentions, and requirements are not characterized well, much less defined. These situations often result when the designer is brought into the product development process too early.

2. **Characteristic-driven design**—Design in a closely controlled situation where strict accountability and proof of adequacy are rigidly enforced. These design situations often involve potentially life threatening situations.

3. **Carry-over-driven design**—Sometimes referred to as "routine" design.

The following classes of knowledge are evident in typical system design practice:

1. Knowledge of basic principles.

2. Knowledge of the general design process in a domain.

3. Knowledge of available components.

4. Knowledge of previous solution approaches.

5. Knowledge of available engineering performance models and workable modeling approaches.

6. Knowledge of test capabilities and results (e.g., what sorts of experimental results and data can be affordably, reliably, or physically acquired).

7. Knowledge of the human network (i.e., where is the knowledge and information in the organization or in professional associations).

8. Knowledge of the requirements, design goals, design decision/evaluation process, and design environment of the current problem.

9. Knowledge of political or governmental constraints.

In summary, design as a cognitive endeavor shares many characteristics with other activities such as planning and diagnosis. But, design is distinguished by the context in which it is performed, the generic activities involved, the strategies employed, and the types of knowledge applied. A major distinguishing characteristic is the focus of the design process on the creation (refinement, analysis, etc.) of a *specification* of the end product.

IDEF6 Language Design Developments

The investigation into language support for IDEF6 included the development of a grammar for describing the rationale for individual design artifacts.

The language is based on the notion that design rationale can be characterized as chains of rationale elements, where each rationale element is expressed in terms of a design artifact and its relation to supporting evidence.

RATIONALE_ELEMENT ---> DESIGN_ITEM, RELATION, SUPPORT

Supporting evidence is expressed in terms of associations between rationale elements and substantiating items.

SUPPORT ----> RATIONALE_ELEMENT, SUBSTANTIATING_ITEM

The kinds of substantiating items that have been identified are as follows:

Elements of a business plan.

Development or operational resources.

. Requirements.

Design goals.

Solution technology.

Solution features.

Business policies.

Natural laws.

Best practice conventions.

Standards.

Laws, regulations, or other social constraints.

Substantiating items are constraints on the design, for example standards and requirements.

SUBSTANTIATING_ITEM ----> SOURCE_MATERIAL_ITEM |
REQUIREMENTS_ITEM |
NOTE | ASSUMPTION |
CORRELATION |
PRACTICE |
STANDARDS |
EXPERIMENTAL_RESULTS |
SYSTEM_CAPABILITIES |
DESIGN_GOAL |
NOMIC CONSTRAINT |
METHOD CONSTRAINT

RELATION ----> satisfies | supported_by

DESIGN_ITEM ----> ARTIFACT | DESIGN_ITEM, S-REL, DESIGN_ITEM

Each artifact may come from IDEFØ, IDEF1, IDEF1X, IDEF3, IDEF4, and IDEF5, as shown in the following grammar production.

```
ARTIFACT      ---->   IDEFØ_ARTIFACT |
                      IDEF1_ARTIFACT |
                      IDEF1X_ARTIFACT |
                      IDEF3_ARTIFACT |
                      IDEF4_ARTIFACT |
                      IDEF5_ARTIFACT
```

The following lists the productions for IDEFØ artifacts:

```
IDEFØ_ARTIFACT      --->    CONCEPT |
                            ACTIVITY |
                            DECOMPOSITION |
                            BUNDLE |

CONCEPT       --->          INPUT |
                            OUTPUT |
                            CONTROL |
                            MECHANISM
```

The following lists the productions for IDEF1 artifacts:

```
IDEF1_ARTIFACT      --->    ENTITY_CLASS |
                            ATTRIBUTE_CLASS |
                            LINK_CLASS |
                            CARDINALITY
```

The following lists the productions for IDEF1X artifacts:

```
IDEF1X_ARTIFACT     --->    ENTITY |
                            ATTRIBUTE |
                            RELATION
```

IDEF1X relation is defined as follows:

```
RELATION      --->          idef1x_relation: RELATION_NAME,
                            RELATION_TYPE,
                            CARDINALITY,
                            ROLE_NAME,

RELATION_TYPE       --->    {identifying, nonidentifying}

CARDINALITY         --->    {MIN MAX} | NUMBER

ATTRIBUTE           --->    ATTRIBUTE_NAME, ATTRIBUTE_TYPE, DOMAIN

ATTRIBUTE_TYPE      --->    {descriptive, alternate_key, primary_key, foreign_key}

ENTITY              -->     ENTITY_ID, ENTITY_NAME, ENTITY_TYPE

ENTITY_TYPE         --->    {independent, dependent}
```

The following lists the productions for IDEF3 artifacts:

```
IDEF3_ARTIFACT      --->    UOB |
                            JUNCTION |
```

I3_RELATION
I3_OBJECT |
I3_STATE |
I3_TRANSITION |
REFERENT |

UOB ---> UOB_NAME, UOB_Identifier

JUNCTION ---> JUNCTION_TYPE, JUNCTION_ID, JUNCTION_LOGIC, TIMING

JUNCTION_TYPE ---> {fan_in, fan_out}

JUNCTION_LOGIC ---> {and, or, xor}

TIMING ---> {asynchronous, synchronous}

IDEF1X relations are defined as follows:

I3_RELATION ---> I3_RELATION_TYPE, I3_RELATION_ID

I3_RELATION_TYPE ---> {precedence, object_flow link, relational}

I3_OBJECT ---> I3_OBJECT_NAME

I3_STATE ---> I3_STATE_NAME, I3_OBJECT

I3_TRANSITION ---> I3_STATE, I3_STATE

REFERENT ---> REFERENT_TYPE, IDEF3_ARTIFACT

REFERENT_TYPE ---> {goto, informational, call_and_continue, call_and_wait}

IDEF4 artifacts are defined as follows:

IDEF4_ARTIFACT ---> FEATURE |

FEATURE ---> OBJECT |
EVENT |
MESSAGE,
METHOD,
ATTRIBUTE,
STATE,
RELATION,
INHERITANCE,
LINK,
TRANSITION,
COMMUNICATIONS_LINK

OBJECT ---> CLASS |
PARTITION |
INSTANCE

CLASS ---> class: CLASS_NAME, (SUPERCLASS)* [FEATURE]*

Using detailed language support would be obtrusive and complicated for the designer. The rationale capture method will need to be better integrated into the natural design process in order to be useable. The development of the language has proved to be extremely useful in understanding the rationale capture problem, and has led to simplifications that resulted in the IDEF6 rationale capture method.

Design rationale should be integrated into the natural design process in which the designer cycles the activities of partitioning, specifying, assembly, simulation, and re-arrangement. During simulation, the designer 'walks' the design through scenarios of use, making note of situations in which the design fails to satisfy constraints and then chooses strategies for satisfying the constraints that have been violated. In this way, the rationale for an artifact in a design can be traced as a series of observations and actions.

These observations were applied to develop a Design Rationale Model for the IDEF4 Object-Oriented Design Method. The Design Rationale Model contains diagrams that describe milestone transformations to design artifacts. Each design situation is represented by a round cornered box with the design state name on the top and a list of diagrams defining the model situation. This strategy allows the designer to choose the level of detail for recording design rationale. The diagrams referenced in a design situation box range from a single diagram illustrating a narrow aspect of the model to all of the diagrams in a design model. If, for example, the designer is making a decision that impacts most of the elements in a partition, then the design situation box could contain all diagrams in that partition. The partition may be used in place of the diagrams if the situation contains all of the models in the partition. Typically, the fewer diagrams contained in a design situation, the more detailed the rationale.

For example, consider the following situation in which a designer captures rationale at a high level of detail. The designer captures the design rationale for the evolution of an *employs/employed by* relation between person class and company class to a link from employee to company. In this case, the "starting" design situation has a reference to a diagram containing the *employs/employed by* relation; and the "ending" design situation has a reference to the link diagram containing the *employed by* link.

Transitions from one design situation to another are represented by an arc with an arrow pointing in the direction of the transformation (see Figure 23). The transition arc lists the observations that necessitated the design change and the changes to be made. For example, in the transition from the design situation containing the *employs/employed by* relation to the situation containing the *employed by* link, the observation will state that the *employs/employed by* relation is not directly implementable, and the action will state that the relation will be replaced by the *employed by* link which can be implemented.

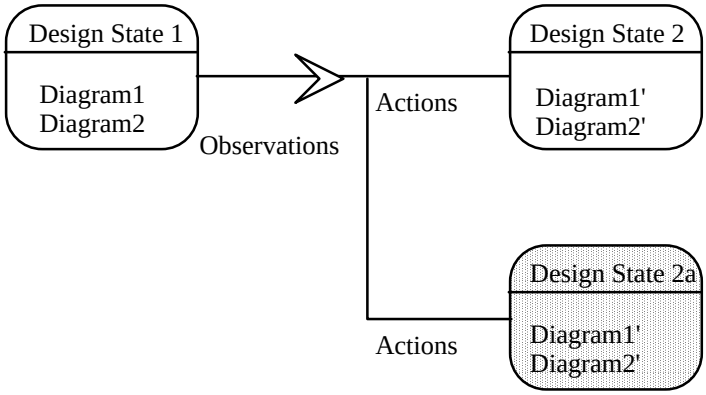


Figure 23.
Observations and Actions Marking Design Transitions

Figure 24 illustrates the concepts of design situation, transition and rationale. In this example, an initial design situation (Design State 1) contains the objects *person* and *company* and the *employs/employed by* relation between *person* and *company*. The use cases show that the access rights of employees and non-employees are different. The designer also notes that the relationship between person and company is conditional because not all people are employed, and that it would be desirable to define this relationship between employee and

company. The designer partitions the object class person into employees and non-employees. This is defined on an inheritance diagram using the subclass/superclass relationship. The designer then redefines the *employs/employed by* relation to be between employee and company and changes its cardinality. These changes result in Design State 2.

In Design State 2, the designer observes that the employs/employed by relation is not directly implementable. There are several options for implementing a relation, including links, arrays, and relation objects. The designer decides to refine the relation to a link. In order to create the link, the designer embeds referential attributes to each class. The link name is written as L1(R1) which denotes that link L1 was derived from relation R1.

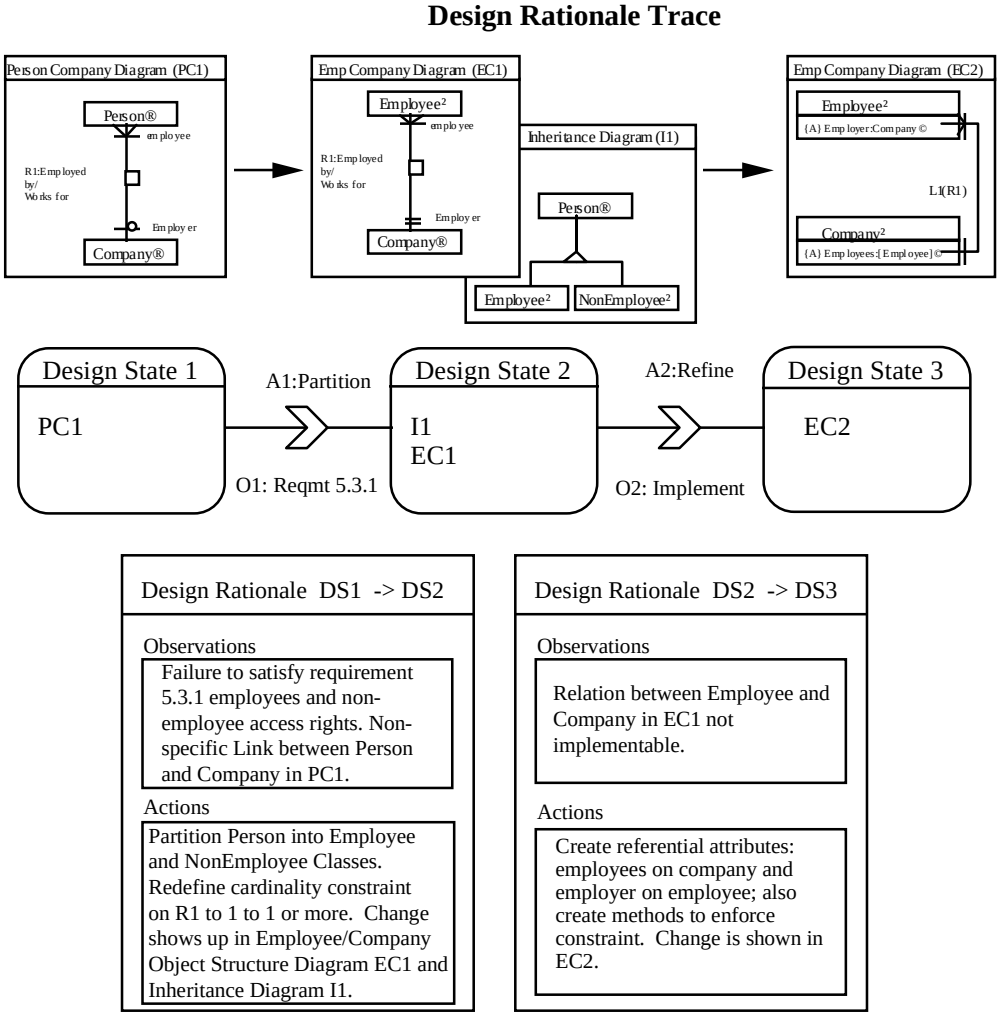


Figure 24.
The Observation/Action View of Design Rationale

Design situations are defined on Design Configuration Forms (Table 4). The configuration form allows the designer to name the situation, give a brief description of the design situation, and list all of the diagrams that are important to that situation. The designer can also identify the design situations from which (Entry Rationale) this design situation evolved as well as the one to which it has led.

Table 4. Design Configuration Specification

Author	*J. Smithe*	**Project**	*IICE*	**Date**	*6/1/94*	**Revision**	*3*
Design Situation		Situation Name		*Situation Name*		ID	*ID*
Description	*Description of design situation*						
Diagrams		Entry Rationale		Exit Rationale			
List of diagrams inactive in this in situation		*List of design entry points in terms of Rationale Specifications*		*List of design exit points in terms of Rationale Specifications*			

Rationale forms record information on design transition, such as why a change was needed and which actions are to be taken. Table 5 shows a sample Rationale Specification Form. The form allows the designer to identify the current design situation, name the goal design situation, record a list of observations that prompted the need for a design revision, and state the actions to be taken. The forms also record the name of the designer, project for which the design is being built, and the date.

Table 5. Rationale Specification

Author	*J. Smithe*	**Project**	*IICE*	**Date**	*6/1/94*	**Revision**	*3*
Rationale		From Design		To Design			
Observation				Action			
List of observations made on design configuration.				List of corresponding actions taken in response to observations made on design configuration.			

Conclusions

Advancement in technology, manufacturing methods, and materials has brought about the emergence of products whose expected usable lifetimes extend over decades and even centuries. Information systems have also evolved from stand alone application-oriented systems with relatively short lifetimes and limited scope toward large scale, distributed systems which must service their users over extended periods of time. Not unlike traditional products, maintenance of information systems whose expected lifetimes may extend over many career periods required explicit capture and storage of the rationale used in their design.

When explicitly captured, design rationale typically exists in the form of unstructured textual comments. In addition to making it difficult to find relevant information on demand, lack of a structured method for organizing and providing completeness criteria for design rationale capture makes it unlikely that important information will be documented.

IDEF6 is designed to capture WHY a design is the way it is, or WHY it is not manifested in some other form, together with HOW the final design configuration was reached. That is, IDEF6 captures WHY, WHY NOT, and HOW a design arrived at its final configuration together with the time-ordered sequence of steps used in the realization of the design. IDEF6 is intended to be a method with the representational capability to capture information system design rationale and associate that rationale with the design models and documentation for the end system. Thus, IDEF6 attempts to capture the logic underlying the decisions contributing to, or resulting in, the final design. The explicit capture of design rationale serves to help avoid repeating past mistakes, provides a direct means for determining the impact of proposed design changes, forces the explicit statement of goals and assumptions, and aids in the communication of final system specifications. Explicit capture of the motivations for why a designer selected or adopted a particular design strategy or system feature for enterprise level information systems is essential to the maintenance of that system over its life-cycle.

Additional work is still needed, however, to realize a fully mature IDEF6 method. The developments achieved to date provide the foundation for future endeavors both to refine and mature the IDEF6 method and to leverage the products of IDEF6 application.

IDEF6 Bibliography

[Dyer 83] Dyer, M. (1983). *In-depth Understanding, A Computer Model of Integrated Processing for Narrative Comprehension*. Cambridge, MA: MIT Press.

[Friel 88] Friel, P. G. (1988). *Modeling Design Reasoning in Automotive Engineering*. Unpublished doctoral dissertation, Texas A&M University, College Station, TX.

[Goel & Pirolli 89] Goel, V., & Pirolli, P. (1989). Motivating the Notion of Generic Design with Information Processing Theory: The Design Problem Space. *AI Magazine*, 10(1).

[Hobbs 86] Hobbs, J. (1986). On the Coherence and Structure of Discourse. In L. Polanyi (Ed.), *The Structure of Discourse*. Norwood, NJ: Ablex Publishing Corporation.

[Kuipers 84] Kuipers, B. (1984). Commonsense Reasoning About Causality: Deriving Behavior from Structure. *Artificial Intelligence*, 24, 169-123.

[Laughton 85] Laughton, J. (1985). *Qualitative Reasoning in Mechanical Design*. Technical Report # XXXX, Department of Computer Science, University of Texas, Austin, TX, 1985.

[Mayer 89] Mayer, R. J., Keen, A. A., & Su, C.J. (1989). *Design Knowledge Management System*. SBIR Phase I Final Report.

[Ramey 83] Ramey, T. L. (1983). *Guidebook to Systems Development*. Internal Research Report, Hughes AirCraft Co., El Segundo, CA.

[Wilensky 83] Wilensky, R. (1983). Planning and Understanding, A Computational Approach to Human Reasoning. Reading, MA: Addison-Wesley.

TOWARD A NETWORK DESIGN METHOD (IDEF14)

Introduction

This report provides an overview of the research conducted toward developing the IDEF14 Network Design Method, a method targeted at modeling and designing computer and communication networks.

IDEF14 can be used to model existing (AS-IS) computer networks or envisioned (TO-BE) computer networks. It helps the network designer play WHAT-IF games with network designs and to document design rationale. The fundamental goals of the IDEF14 method research project have developed from a perceived need for good network designs that can be implemented quickly and accurately.

What is Network Design?

Digital computer networks provide the technology for the transportation of information across geographical areas. Today, computer networks have permeated industry, government, and education, and have created new communities in which people from geographically diverse regions interact frequently. Computer networks have started the information revolution of this century.

In an enterprise, the role of computer networks cannot be overstated. The computer network links 'islands of information,' and allows information to be readily accessible at different points of the enterprise's presence. Computer networks are part and parcel of the information infrastructure of an enterprise. They form the enabling technology for new paradigms of enterprise computing such as client-server, workflow, information warehousing, and electronic commerce. As enterprise information and its integration become more important to a modern enterprise, its computer network must support the quality, accuracy, effectiveness, and timeliness of the distribution of the information.

Like any other component of an enterprise, the computer network undergoes continuous changes. These changes are brought about by two broad factors.

1. The requirements of the services that are supported by computer networks change.

2. Rapid progress in network technologies allow for more efficient, more reliable, and cheaper solutions for networking requirements. This, in turn, has led to the availability of new communication services.

An optimal solution to a network design problem for an enterprise is governed by the following objectives [Minoli 93a]:

1. Minimize the cost (i.e., build the cheapest network possible).

2. Maximize the cost-performance ratio (maximize Return on Investment).

3. Maximize the Quality of Service (QOS). For example, lower the turnaround time for a network service.

4. Minimize risk of loss (maximize reliability).

5. Maximize the enterprise's profits (e.g., reach new markets through national and global networking).

6. Maximize the growth opportunity (i.e., the flexibility) of the network.

7. Maximize the prestige of the firm.

8. Maximize work force productivity.

The above objectives are not orthogonal to each other. However, a well designed network must support most of the objectives. Also, different enterprises place a different importance on each objective. For example, maximizing the reliability is a key objective in the design of a military network.

The following are the inputs or constraints to a network design process:

1. Locations of demands for services.

2. Functionality or services to be considered.

3. Choice of network architecture; this includes service availability for a specified area.

4. Cost and tariff information for network components and services.

There is a need for a formal method for capturing the above inputs in order to carry out network design. The method must support both the creation of alternative designs and the ability to obtain various figures of merit of the design.

What is IDEF14?

IDEF14 is a method that supports the design of computer networks by allowing the designer to capture requirements, specify network components, capture the existing network configuration, and perform analyses on the design. It also provides support for managerial decision-making and engineering economy. Finally, IDEF14 provides the ability to represent and store design rationale of network designs.

In Zachman's pioneering work on information systems architecture [Zachman 87], he states that the network model is an independent, yet complementary, perspective in the Framework for Information Systems Architecture. A network can be modeled at different levels of abstraction. These levels of abstraction correspond to the rows of the Framework.

Motivation for a Network Design Method

Most enterprises have a network which is enhanced and modified over time. Enterprises rarely replace their entire computer networks; more often, an existing network is enhanced, or part of the network is replaced. Hence, it is important to capture the 'AS-IS' computer network. This involves the ability to represent the existing layout of the network and the specifications of existing network components.

To support the TO-BE network, the method must have the capability of representing service requirements. It must also be able to represent the different technologies available for each component or communication service of a network. This includes representing the technical specifications that are required to obtain the analyses of the overall design. This also demands representation of the costs and tariffs of the component or communication service. IDEF14 addresses the following problems and needs associated with network design.

Layout Specification, Decomposition, and Topology of a Network

A network can be modeled at any level of detail. At the coarsest level of detail, a network is simply a single node. That node can then be decomposed according to a level of detail, say according to geographical distribution across cities. Each node of the detail can be further decomposed until one reaches a point where further decomposition yields nothing useful.

Computer networks are generally classified as Local Area Networks (LANs), Metropolitan Area Networks (MANs), and Wide Area Networks (WANs).

IDEF14 must be able to represent the topology of the network at various levels of decomposition. For example, IDEF14 must be able to represent the topology of the network of an enterprise that is geographically distributed. At the same time, it must represent the part of the network within a city, building, even a room.

IDEF14 must be able to represent both regular and irregular topologies at any level of decomposition. Regular topologies are usually used in LANs and MANs; these include star, bus, and ring topologies. A non-regular topology is constructed by using many point-to-point links to join the nodes of the network.

Workload and Quality of Service Specification

The workload represents the characteristics of the burden contributed by a request center on the network. Quality of Service (QOS) represents the minimum requirements of a request center. Examples of QOS are as follows:

1. Average response time must be not more than 2 seconds.

2. End-to-end blocking probability must be not more than 0.005.

Behavior of Network Elements

These are the qualitative characteristics of a network component. The following are examples of behaviors of a network element.

1. Routing traffic on alternate routes.

2. Queuing disciplines at various nodes.

3. Access protocols for multiple-access channels.

Characteristics of Network Elements

Examples of quantitative characteristics of a network component include:

1. Failure rate of a bridge.

2. Maximum bandwidth of a bus or a point-to-point link.

3. Number of ports in a hub or bridge.

Cost and Tariffs

The IDEF14 method must be able to represent cost information, including fixed and variable cost of each component of the network. Since an enterprise network is a combination of private elements and public (switched) services, the tariffs of potential services must also be represented.

Languages for IDEF14

Computer networks traditionally have been modeled as graphs. Most aspects of computer networks related to topology and reliability possess a graph theoretic basis. While it is clear that the network design methods have a graphical language, there is a need for representing and manipulating constraints that exist in computer networks. IDEF14 must address this issue.

IDEF14 Products

The product of an IDEF14 application is a network design expressed as a configuration of network components and their specification (protocols, bandwidth, etc.), along with other models of the design (i.e., queuing model, reliability model, cost model) and a design rationale document of the design.

Users and Key Beneficiaries of IDEF14

There are several types of individuals who will be involved with the IDEF14 method. These classes are listed below:

1. *Managers* are ultimately responsible for sanctioning the changes to the network. The method will help managers understand the requirements, and evaluate alternative solutions and their estimated performances and costs. Also, the design rationale component of IDEF14 method will help managers understand the rationale behind a recommended network design.

2. *Modelers* are responsible for generating the IDEF14 AS-IS computer model of the enterprise. Modelers are generally individuals of the network management division of the enterprise. Modelers are experts in the use of the IDEF14 method, and are responsible for generating the necessary models and for performing existing network analyses.

3. *Designers* are responsible for identifying and evaluating solution alternatives for the design of the new computer network. They are members of the development team who create a plan for implementing the new computer network. Network designers are experts in the use of the IDEF14 method. They must analyze requirements, develop alternate solutions, evaluate alternate solutions in terms of performance and cost, document design rationale, and finally select/recommend the most appropriate design for implementation.

4. *Reviewers* are responsible for examining the AS-IS network model generated by the modeler. This function is designed to help modelers maintain consistency between the existing network and the model of the network. The reviewers are also responsible for validating new designs.

5. *Implementors* are responsible for taking the IDEF14 network design and implementing it. These are members of the design team who are involved in the physical realization of the system from the abstract descriptions and models. They must prepare the design for implementation, participate in the purchase and installation of the various components and services, test the new network, and verify that the requirements have been met.

Benefits

IDEF14 involved the creation and generation of the models shown in Figure 25. The Network Configuration model is a graphical model that captures the topology, configuration, specification, and attributes of network components. Using this model, the Queuing, Reliability, and Cost models are generated. These three models are analyzed and used as input to the decision making process of network selection.

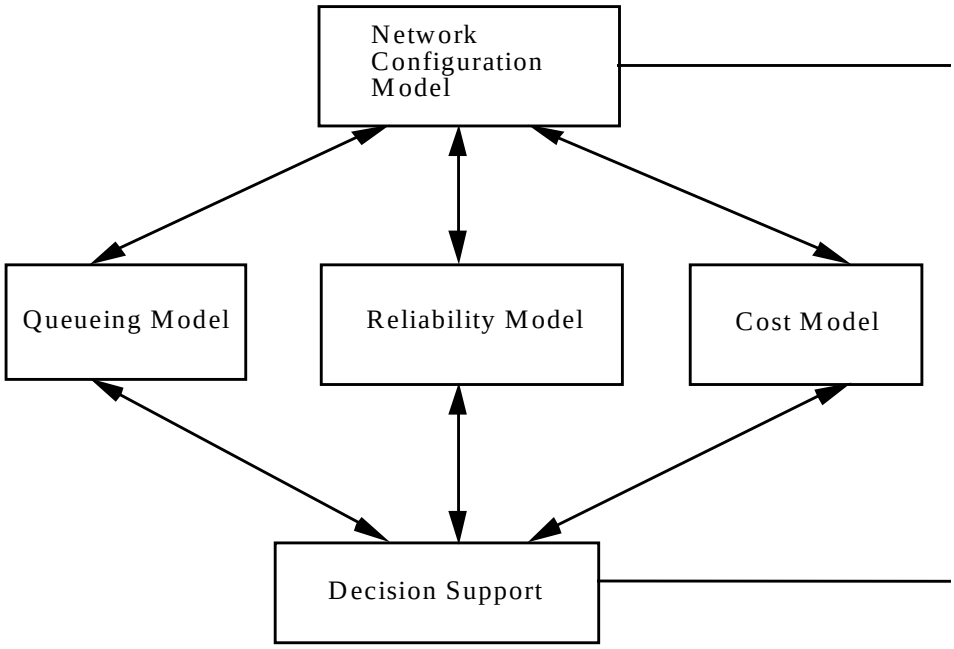


Figure 25.
Models Generated Using the IDEF14 Method

The following are some benefits that IDEF14 provides for modeling and designing computer networks.

1. The IDEF14 method helps a modeler build useful network configuration models of AS-IS networks. The network configuration model also yields other models for analyzing the AS-IS network.

2. The method provides support for obtaining Queuing, Reliability, and Cost models of alternative network designs (see Figure 25). Analyses performed on these models are used in the decision support activity of the IDEF14 method. The IDEF14 method will help those who possess a fairly strong understanding of computer and communication networks make good network designs. Thus, network engineers who are not necessarily expert network designers can design networks using a tool based on the IDEF14 method.

Summary of Developments and Research Findings

IDEF14 Basic Concepts

A complete understanding of IDEF14's basic concepts is needed to effectively apply the procedural and language components of the prototype method. Fourteen basic concepts serve as the foundation of the IDEF14 method:

Node	Subnetwork
Link	Topology
Workload	Protocol

Queuing Theory	Reliability Analysis
Quality of Service	Cost Analysis
Decision Support	Design Rationale
Network Constraints	Component Libraries

The purpose of this section is to describe these concepts and provide examples of each.

Node

A node in the IDEF14 method represents a piece of hardware that is part of the network. Examples of nodes include personal computers, file servers, workstations, mainframes, terminals, routers, bridges, repeaters, gateways, switches, Very Small Aperture Terminals (VSATs), or any other hardware unit that is connected to the network. All node types share common attributes. For example, a name must be allocated to each node, irrespective of its node type. However, there are attributes that differ among node types. For example, a switch has *number of ports*, as would a parameter.

Subnetwork

A network can be decomposed into one or more subnetworks. There are two types of subnetworks that must be modeled by IDEF14:

1. Networks or subnetworks of the enterprise that have decompositions. For example, a global enterprise may have networks in each of the countries where they have a business presence. At the highest level of modeling, the networks corresponding to each country form a subnetwork.

2. Networks or parts of networks that are public or are part of a service (e.g., the Internet and value-added networks). These are subnetworks that the modeler may not want to model in detail; hence, this subnetwork is not decomposed further. Since there is ultimately a connection between nodes of the enterprise network and a node of the subnetwork, the subnetwork node making the connection is represented in IDEF14.

Topology

Most modern LANs and MANs have a fixed topology around which the computers are configured. The topology concept in IDEF14 is intended to represent standard topologies for LANs and MANs. Examples of topologies include star, bus, ring, dual bus, dual ring, etc. Attributes of topology are the access medium (whether twisted pair or coaxial cable or fiber optic cable) and the maximum bandwidth of the medium.

Link

In IDEF14, a link represents a point-to-point communication medium over which data and voice transmission takes place. Examples of links are twisted pair, wireless link, satellite channel, T1 and fractional T1 lines, coaxial cable, fiber optic line etc. As an example, common attributes of all links are maximum bandwidth.

Protocol

This concept corresponds to the Link Level (Level 2 of the OSI Reference model) protocol. This is a is a feature of the link or the LAN. Examples of protocol include Serial Link Level Protocol (SLIP) and Point to Point Protocol (PPP) for links. Examples of protocols for LANs include the IEEE 802.3 CSMA/CD protocol

and the IEEE 802.4 Token Bus protocol both of which run on a bus topology. In the case of a point-to-point link, both the nodes adjacent on the link must run the protocol of the link. In the case of a LAN, all the nodes connected to the LAN must run the protocol of the LAN.

Workload

For data networks, the workload attributes typically are the number of packets generated or transmitted per second and the arrival distribution. Workload can be further decomposed into classes. As an example, the IP packets generated by a computer can be further classified as TCP, UDP, IGMP, etc. As another example, traffic generated by a computer on a Fiber Distributed Data Interface (FDDI) LAN must be classified as synchronous or asynchronous. The workload attributes (arrival rate and distribution) must be specified for each of the classes that comprise the entire workload. Because workload varies during the course of a day, it is necessary to specify workload as a function of the time of day. This can be specified as the average workload during the first hour (12:00 a. m. - 1:00 a. m.), the second hour, etc. Often, workload must be specified within a time interval. In this case, the interval of time becomes part of the workload specification.

Queuing Model

Queuing theory [Klein 75] is used to analyze the performance of computer networks. Typical performance measures are:

1. End-to-end delay. This measure, in turn, is used to obtain the average turnaround time of a service request.

2. End-to-end throughput.

3. Blocking. This is a measure of the unavailability of a required resource or service.

4. Data Loss. This is especially true of broadband networks. Examples of data loss include cell loss in ATM networks, frame loss in frame relay, and protocol data units (PDUs) loss in Switched Multimegabit Data Service (SMDS) networks.

Analysis obtained by queuing analysis helps in selecting the most feasible design among various alternatives. Analysis of queuing models for computer and communication networks is well founded [Klein 76, Schwartz 88]. Queuing theory is used to capture the qualitative and quantitative aspects of each component of the network that are needed in the queuing analyses. This includes the following:

1. Specification of the different classes of packets. For example, in an ATM network, there are four different classes of services. A cell arriving at an ATM switch belongs to one of the four classes.

2. For each service class, the packet/cell/frame arrival rate. This is part of the workload characterization of the nodes that generate traffic.

3. For each service class, the packet/cell/frame arrival distribution. This is part of the workload characterization of the nodes that generate traffic.

4. The queuing discipline (prioritized, first come—first served, etc.) at each queuing center.

5. The service distribution at each queuing center of the network.

6. Maximum length of each queue. This is related to the maximum buffer space and the average length of the packets.

Reliability Model

Reliability analysis is done on a network to analyze the performance of computer networks. Typical performance measures are [Colb 87]:

1. 2-terminal reliability. This is the probability that a path exists between any two specified nodes of a network.

2. All terminal reliability. This is the probability that for every pair of nodes in the network, a path exists between the pair.

3. k-terminal reliability. This is the probability that for **k**>2 specified nodes of the network, a path exists between any two of the **k** specified nodes.

In general, the reliability of a network is enhanced by using superior components and by having redundancy of components along the critical paths of the network. Because both of these solutions involve a higher cost, reliability analysis is necessary in the decision support of the design selection. The scope of IDEF14 method in supporting the reliability analysis of a network is to capture qualitative and quantitative attributes of various components of the network that are needed for the reliability analysis. Examples of these attributes are probability of failure and mean time between failure (MTBF) for each component of the network.

All three types of reliability measures mentioned above are hard problems for general graphs. Algorithms relating to exact and approximate reliability analysis can be found in [Colb 87].

Quality of Service

IDEF14 provides support for specifying the quality of service (QOS). The QOS can be specifiable at component, subnetwork or the entire network level. It can be related to performance (e. g., end-to-end-delay must not exceed 10 seconds) or to reliability (2-terminal reliability between point A and point B must be at least 0.995).

Cost Model

The cost model is used as part of the decision making process for network design selection. The scope of the IDEF14 method in supporting the cost analysis of a network is to capture qualitative and quantitative attributes of various components of the network that are needed for the cost model. Examples of these include fixed costs, cost of maintenance, expected life of equipment, and so forth. Cost models for economic decision-making can be found in [Riggs 82, Newnan 80].

Decision Support

IDEF14 is intended to allow network designers to arrive at different network design alternatives, each with different characteristics. There is a need to support the process of selecting a design for implementation. Decision support must include at least a mechanism to display the results of the analysis performed on the Queuing, Reliability, and Cost models of each design, and the rationale behind each design.

The designer must be able to assign weights to the Performance, Reliability, and Cost measures in order to select a design for implementation.

Design Rationale

The design rationale component of the IDEF14 method facilitates the acquisition, representation, and manipulation of the design rationale used to develop IDEF14 designs. *Rationale* is the reason, justification, or motivation that moved the designer to select a particular design feature. Simply put, design rationale focuses on why the design is the way it is, from the designer's point of view.

The scope of the rationale component must cover all phases of the IDEF14 development process, from initial conceptualization through detailed design and evaluation activities, to final selection of a design for implementation.

Network Constraints

A network design must obey constraints that exists in computer networks. As an example, a frame relay service provider at a location may provide services only on T1 links (and not on fiber optic). Other examples include the fact that both adjacent nodes on a link must run the protocol of the link.

Network constraints must not be confused with design constraints. Design constraints are basically the constraints that the network design must satisfy. Design constraints are expressed in terms of quality of services, maximum costs, and so forth.

Component Libraries

With computer and communication technologies advancing so rapidly, there is a need for the IDEF14 method to accommodate new technologies. It is important to have libraries for various concepts of the IDEF14 method. There will be a library for each concept. Each concept library will contain instances of the concept that are well understood and prevalent. In addition, relevant technical information pertaining to an instance of a concept will also be stored in the library. As an example, a router will be an instance of the node library. Technical specification of routers such as rejection and pass-through rate are part of the attribute list of the node type *router*.

Users can add new instances to each concept library. For example, a user can add a new topology to the topology library, should a new LAN have a topology that is not part of the topology library.

IDEF14 Procedure Developments

This section presents a prototype procedure for network design, validation, and refinement.

There are three scenarios of use where the IDEF14 network design method is useful. They are:

1. Upgrading an existing enterprise network (most general case).

2. Modeling the AS-IS network of the enterprise.

3. Designing a network from scratch.

In this section, we present a prototype procedure for network design, validation, and refinement. We focus on the most general case (i.e., upgrading an existing enterprise network). For this case, the general IDEF14 procedure comprises the following three broad tasks.

1. Capture the existing network.

2. Capturing the workload of the TO-BE network.

3. Designing the TO-BE network.

The second and the third scenarios are subsets of the first. For the second scenario of use, modeling the AS-IS network, only the first broad task is required. On the other hand, the third scenario of use requires that the second and third broad tasks be performed.

The procedure presented in this section assumes a large network design effort involving a team approach. Projects that are narrower in scope may not require all the activities described below. As with all methods, the application procedure depends largely on the purpose for which the method is being used.

Network design is an evolutionary process through which existing networks are modeled, requirements are gathered, and designs are created, evaluated, validated, documented and refined. In general, when IDEF14 is used to design networks, the following activities or steps are applied recursively:

1. *Collect* - Acquire knowledge for existing network components, for requirements, and for existing technologies.

2. *Classify* - Individuate network concepts (meta types) such as node types, link types, topology types, as well as elements of types.

3. *Hypothesize* - Postulate candidate network designs from the data and evidence acquired.

4. *Validate* - Ensure that AS-IS network model is correct and the network constraints in designs have been met.

5. *Analyze* - Generate queuing, reliability, and cost models of the network design, validate the models, and analyze the models.

5. *Challenge* - Involve domain experts in testing the conclusions of designers as to the validity of their conclusions.

6. *Refine* - Filter, improve, adjust, and add detail to the network design.

These steps are embodied in the prototype IDEF14 procedure presented later in this chapter. The IDEF14 activities should be considered "modes of thought" rather than sequential steps. Designers should not expect to apply these activities in a strictly sequential manner. Nor should they expect that organizing activities by project phases necessarily defines when those activities start or stop. Rather, phases reflect which modes predominate during a given interval of time. Thus, modes of activity may be organized into phases to assist with management of the project. The following section describes the modes of activity constituting the procedure for IDEF14, thus establishing a basic framework for the network design effort.

Mode Zero: Define the Project

The network design team must establish the purpose and scope of the network design effort as early as possible in the project. The purpose statement provides a "completion criteria" for the network design effort. The purpose is usually established by (1) prioritized objective statements for the effort, (2) statements of needs that the network design effort must satisfy, and (3) questions that the client wants answered. The scope of the project is established by a set of statements that bound the area addressed by the project. For example, the network design may involve upgrading a subnetwork of the enterprise network without modifying the rest of the network. Thus, scope statements identify the specifically targeted areas of network design activity and identify those that are explicitly ignored.

The purpose and scope can rarely be determined completely in advance. The client often revises his list of questions as the data starts being compiled. The area that an analyst thinks will lead to the answer often leads in other areas that were not originally considered. The purpose and scope generally evolve during the initial part of the project. The purpose and scope of an IDEF14 effort are captured on an IDEF14 Project Summary Form similar to the one shown in Figure 26.

IDEF14 Project Summary Form

Project Title: _____

Project Leader: _____

Purpose: _____

Major in-scope subnetworks: Major out-of-scope subnetworks:

Figure 26.
IDEF14 Description Summary Form

Define the Purpose

Defining the purpose is an important initial step in the network design effort. If the purpose is taken for granted or ignored, project personnel are likely to find their efforts ignored by the client. Without a purpose statement, the only completion criteria are budget and time. Conversely, with a regularly reviewed and clearly defined purpose, the project can often be completed under-budget. Defining the purpose involves listing the stated objectives of the client and the specific source(s) of each (e.g., person, project, or organization), defining the information goals of the project in terms of how the network design will be used, and establishing priorities among the stated objectives and information goals of the effort. The process of developing a purpose statement can be facilitated by involving the client in answering questions like the following:

1. What problematic symptoms, concerns, or opportunities are of the greatest interest to the client?
2. Who will be the beneficiaries of the newly implemented network?
3. What questions does the client need answered?
4. What issues are behind the need for network design?
5. What decisions are involved in the selection of a network design?

Establish the Scope

Once the purpose of the effort has been characterized, it is possible to define the scope of the project. Defining the scope begins with delineating the subnetworks of the enterprise on which the network design effort is to be performed and documenting them in a set of scope statements. Ideally, scope definition should identify only those areas that are relevant to the needs of the client. However, in a network, changes to one subnetwork often affect other subnetworks of the enterprise network. Hence, out-of-scope subnetworks must also be defined.

Mode One: Develop Team and Organize for Data Collection

Once the initial project purpose and context have been determined, it is necessary to organize for data collection. There are three activities in this phase of network design: solidifying the makeup of the project team, assigning roles to team members, and assigning scenario development responsibilities to team members.

The following roles are normally assumed by personnel involved in a network design effort.

1. *Designer*: The IDEF14 expert who will be the primary developer of the IDEF14 network design models.

2. *Client*: The person or organization requesting the network design effort development.

3. *Domain expert*: The knowledge source person in the network department of the client.

4. *Primary contact*: The individual who acts as the interface between the analyst and the domain expert.

5. *Project leader*: The person ultimately responsible for the entire network design effort.

6. *Reviewers*: Persons knowledgeable in the domain and/or the IDEF14 method responsible for reviewing and approving draft models and documents. Reviewers authorized to make written critiques of IDEF14 designs are *commentators*. The remainder are *readers*. Both team members and domain experts can be reviewers.

7. *Librarian*: A person assigned the responsibility of maintaining source material logs and files of documents, making copies, distributing kits, and keeping records.

8. *Team members*: All personnel involved with the IDEF14 network design effort.

For large projects, the role of project librarian is essential. In smaller efforts, that role may be assumed by the analyst. In establishing the librarian function, the project leader assigns an individual to be responsible for collecting, cataloging, controlling, and distributing source material, kits, glossaries, files, and so forth throughout the project. Additionally, the librarian is responsible for assembling reference models and materials from external sources that can be used to accelerate team efforts. The librarian may also maintain a glossary of terms as a reference to ensure that analysts understand terminology. Whether maintained by the librarian, or informally shared among analysts, the glossary of terms will grow and undergo incremental refinement throughout the project.

A pivotal task in organizing the data collection effort is identifying the key sources of domain knowledge and information. Working with the primary contact, the project leader or analyst compiles a list of experts to be interviewed. In compiling this list, it is helpful to obtain background information about each expert. This includes information about the responsibilities, current assignments, and other areas within or related to the domain in which the expert has experience. The name, location, and telephone number of the experts should also be recorded.

Throughout the data collection effort, other valuable sources of information will be identified. Some of these include operating instructions, procedure manuals, employee handbooks, regulations, policy manuals, project files, reusable IDEF models, and models derived with other methods and techniques. These items often constitute evidence of constraints or provide references to evidence in the domain. As an example, network failure reports help establish requirements related to the reliability of the TO-BE network.

In addition to organizing the structure of the team, the project leader also organizes the activities of the team. Organizing network design activity may begin by casting the general IDEF14 procedure into a method application guide tailored to the specific needs of the project. A method application guide outlines a project-specific application of the IDEF14 procedure and is used by the analysts on the network design team. The method application guide includes standard outlines for interviewing domain experts, method and tool interface

specifications, project library use procedures, and a standard glossary of terms. This guide may be accompanied by a project plan. A typical project plan will delineate phases of effort with clearly established tasks and milestones, intermediate and final deliverables, individual team member assignments, informal and formal reporting structures, and so forth.

Mode Two: Collect and Analyze Information

With the team organized and the approach for network design outlined, the team begins collecting evidence. By direct interaction with personnel of the network and EDP departments, network design team members document observations and collect information relating to the existing networks and requirements of the new network; this information will later be analyzed and used as the basis for arriving at network design.

Prepare for Interviews

The most valuable mechanism of evidence collection is the interview. Interviews with corporate communication/network personnel provide an opportunity to collect information relating to the configuration and specification of the existing network and problems relating to existing network. Interviews with network personnel and network users provide an opportunity to collect requirements of new services and quality of services (QOS) for the TO-BE network.

While the specific interviewing approach and format are likely to vary across projects, some guidelines are recommended. Before the interview, the analyst should prepare a tentative agenda and some specific questions. Analysts are encouraged to prepare a brief outline of: the purpose of the interview, the topics to be covered, the types of information being sought, the authority for requesting the interview, and probing questions that can be used to motivate discussion. On large projects, project leaders may wish to include more formalized interview preparation guidelines and standards in a method application guide, including standard interview planning sheets, question templates, glossaries of terms, and so forth.

The ultimate success of the interview depends largely on the preparation made by the analyst. A number of activities contribute to successful preparation:

1. Schedule the interview and make necessary logistics preparations.

2. Establish the goal(s) of the interview.

3. Prepare candidate questions.

4. Anticipate the probable questions and concerns of the person being interviewed and be prepared to resolve concerns.

Once a list of experts to be interviewed has been compiled, an interview schedule can be developed. Interviews are normally scheduled with domain experts through the primary contact. The analyst should make sure that the scheduled time and duration of the interview is coordinated with the person being interviewed and his or her supervisor. Additional logistics considerations are also important to the success of the interview, such as reserving a suitable location to conduct the interview and arranging for the necessary supplies.

The goal(s) of the interview should also be established up front. In establishing the interview goals, analysts establish why the interview is being scheduled and what information is needed from the domain expert. Preparing a succinct goal statement is often helpful to provide a general direction for the interview line of questioning.

Once the goal(s) of the interview has been established, candidate questions can be formulated. Candidate questions should be written down and organized into a logical sequence. Candidate questions should be clear, use words and phrases appropriate to the background of the person being interviewed, and invite rather than lead answers. In preparing candidate questions, it is often useful to explore the following broad topics:

1. What are the configuration and specification of the existing network?

2. What are the problems associated with the existing network?

3. What are the pressing problems that the person being interviewed would like eliminated or minimized in the new network design?

4. What are additional features that the person being interviewed would like in the new network design?

An element of preparation often overlooked by inexperienced analysts is the need to explain why the domain experts are being interviewed, what will be done with the information they provide, and what they can expect in return. Each interview should establish a mutual understanding of these items before attempting to satisfy the information needs of the analyst.

Interview Domain Experts

Interviews may be conducted throughout the project with one or more of the following goals in mind:

1. To collect additional information.

2. To confirm and/or clarify previously collected information.

3. To validate IDEF14 descriptions with the domain expert.

4. To obtain leads for acquiring additional information.

Collect and Catalog Valuable Sources of Information

When appropriate, designers should request copies of existing documents, operating instructions, manuals, network models, designs that are part of the existing network, network failure reports, logs of problems, etc. These sources could be either in electronic or paper form. If information sources are in electronic form, the design team must obtain (read) permission to access the information.

Analyze Collected Data

Following data collection, interview notes are compiled and the initial findings are cataloged into lists called pools. Node and Link and Topology are meta network component types because they contain different network component/topology types, with each type having distinct attributes. Pools will exist for each type. The following two processes are enacted iteratively.

1. Create node (computer, terminal, bridge, switch, etc.), link (twisted pair, coaxial cable, microwave link, etc.), and topology (bus, ring, dual bus, etc.) types.

2. Populate the pools corresponding to the existing types with the data collected. In some cases (especially for nodes and links) the populated pools will include product data and specifications of commercially installed equipment and commercially available services.

Mode Three: Model AS-IS Network and Workload Characterization

At this point, a pool of elements that are needed for modeling the AS-IS network has already been created. The designer should use the graphical language to model the AS-IS network, using elements from the pools. Also, the workload characterization of each service center must be modeled.

The model of the existing network must be validated. The validation should first be done by the team members and then by the design experts. Once the AS-IS network model has been validated, the elements and their attributes must be marked in order to distinguish them from new elements that will be in the TO-BE models.

Mode Four: Update Pools with Existing Technology

The network designers enhance the pools with components and services that will potentially be used in the TO-BE designs. Sources for updating the pools are diverse. Primarily, new components and services that are added to the pools are reflections of the experiences and expertise of the design team. In general, information relating to new components comes from product catalogs, magazines related to network and information systems, relevant news groups, and by word of mouth. Information relating to new and unused services comes from local and long-distance telephone carriers and from data networks that service various locations in which the enterprise has a presence.

It is necessary to store as much information (in terms of attributes of pool elements) about technologies that are being used or have potential for the enterprise network. This is necessary to generate the queuing, reliability, and cost models of the various designs.

Mode Five: Design the TO-BE Network

This mode includes the steps taken to arrive at different network designs and the selection of the most appropriate design. The first step in this mode is the creation of a new design or the modification of an existing design. Based on the outcome of some later steps, it may be necessary to go back to the first step. The processes involved in this mode are shown in Figure 27.

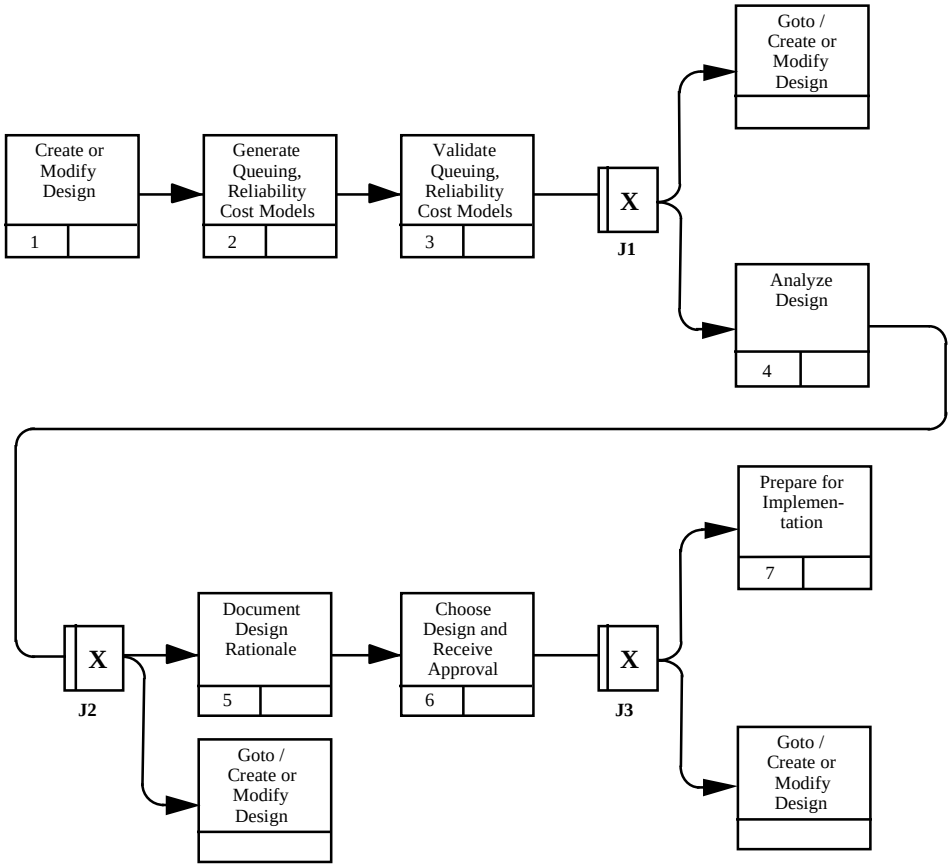


Figure 27.
Process Description of Mode Five of the IDEF14 Procedure

Create First Cut Design or Modify Existing Design

The designers use their expertise to create a first cut design, or modify an existing design. As part of this step, new components or services are added from the pools or are used to replace existing components or services. It is necessary that the additions or changes meet the network constraints. Also, all attributes of the components/services in the design must be specified in order to arrive at proper Queuing, Reliability, and Cost models.

Generate Queuing, Reliability, and Cost Models

The Queuing, Reliability, and Cost models must be generated from the network design under consideration.

Validate Network Design and Queuing, Reliability, and Cost Models

As a first step to validation, the designers must ensure that network constraints have been met by the design. The Queuing, Reliability, and Cost models that were generated must then be validated. An invalid Queuing, Reliability, or Cost model could result from incomplete or missing attributes of one or more component/services that are part of the design. A network design yielding an invalid queuing, reliability, or cost model needs to be modified.

Analyze Designs and Verify Analysis

The Queuing, Reliability, and Cost models must be analyzed to obtain the performance, reliability, and cost characteristics of the design. The analysis must be verified by members of the design team who are not the analyzers of the design. In the case when the analysis indicates that the design is a potential design, the design team must proceed with documenting the rationale behind freezing the design. If the analysis does not satisfy most of the design constraints, then the design needs to be scrapped or modified.

Document Design Rationale

It is necessary to document the design so it can be referenced, particularly by members of the design team and by implementors.

Select Design

It is assumed that a number of alternative designs will be available for selection. The selection is a non-trivial task. This is because the network must satisfy various design constraints pertaining to performance, reliability, and cost. If it is found that none of the network designs are suitable for implementation, then a new design needs to be created. This could result in either starting a design that is radically different from the others or could result in a modification of one or more existing designs.

Get Approval for Implementation

A selected design has to be approved by the manager of the financial department of the enterprise. In making a case for the approval of the design, the project manager has to present the design, the results of its analysis, and the rationale behind the design. If it is found that none of the network designs is suitable for implementation, then a new design needs to be created.

Prepare for Implementation

Once a design has been selected and approved, it must be prepared for implementation. This step involves describing the design and its rationale to the implementors of the design.

Mode Six: Implement the Design

Once a network design is selected and approved, it has to be implemented. Although the actual implementation of the network design is beyond the scope of the IDEF14 procedure, new problems or requirements not previously detected may occur during implementation. In such cases, the design must be modified to account for the new problems or requirements.

IDEF14 Language Design Developments

Computer networks traditionally have been represented as graphs, where a node of the graph represents a piece of hardware such as a computer, and edges of the graph represent communication links. It is necessary that IDEF14 have a graphical language, based on graph layout. In this section, we describe the graphical language of IDEF14, and justify a need for a textual language.

Graphical Language Design Developments

In order to distinguish among different types of nodes, each type of node is represented by a different artifact. Standard node types must have predefined icons for representation. In the case of user-defined type, the user must specify a different icon to represent the new type. The graphical representation must allow for annotating various attributes of the node close to the node.

In IDEF14, a subnetwork is represented by a cloud. For subnetworks of the enterprise, the corresponding cloud representation has a decomposition. For example, a global enterprise may have networks in each of the countries in which it has a business presence. The network of each country is represented by a cloud that has decompositions corresponding to more detailed model of the subnetwork. In the case where networks or parts of networks are public or are part of a service, the cloud is not decomposed further. Because there is ultimately a connection between nodes of the enterprise network and a node of the 'cloud,' the node of the cloud is represented as a node artifact within the cloud.

In the case of links, each link type is represented by a different artifact. Like in the case of new user-defined node types, the user must specify a different icon to represent a new link type. Attributes of a link must be annotated on the diagram itself, close to the link.

Topologies are represented by figures that closely represent the physical topologies of the network. For example, a ring topology is represented by an oval and a bus topology is represented by a line segment.

Workload is represented by a circle. A workload generated by a node is drawn close to the icon corresponding to that node.

An example of the graphical language used to model an enterprise network is shown in Figure 28. The enterprise is located in two cities: Houston and New York. The Houston location has a LAN called LAN_4 which is an Ethernet LAN. A computer called Egret has its workload specified. The New York location has three LANs. LAN_1 is a 16 port, 10BT Ethernet LAN with Simple Network Management Protocol (SNMP) capability. LAN_2 is a Token Ring LAN connecting three computers. LAN_3 is a dual loop FDDI LAN that has connections to both LAN_1 and LAN_2 via Router3 and Router4, respectively. The Houston and New York subnetworks are connected by a Frame Relay service. The Houston subnetwork connects to the Frame Relay cloud via Router1 and a 256 kilo-bits per second (kbps) leased line, whereas the New York subnetwork connects to the Frame Relay cloud via router Router2 and a full T1 leased line. At a coarser level of detail, the Houston and the New York subnetworks could each be represented as a cloud.

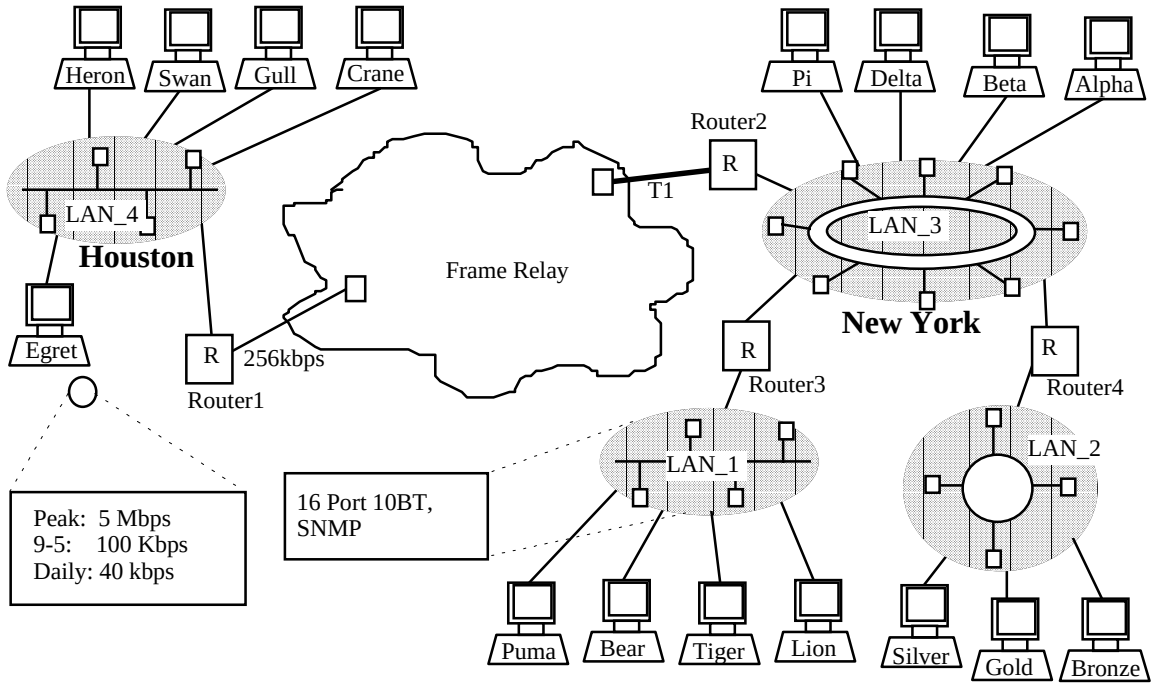


Figure 28.
Example of an Enterprise Network

Textual Language Design Developments

A graphical language is not expressive enough to represent network constraints. In addition to network constraints, there is a need to express characteristics (of network components) that are inherently mathematical in nature. For example, the average service time for an ATM switch to route an ATM cell is likely to be a complex set of mathematical equations. There is a need for a textual language to support this. We anticipate that the IDEF14 textual language will be similar to the textual language of the IDEF5 Ontology Capture method [KBSI 94].

Summary of Accomplishments

In this report, we have demonstrated the need for the IDEF14 design method. We identified the various components that should (and should not) go into the IDEF14 method. We outlined the broad steps and models that are part of a network design effort. We provided the basic concepts that are needed for the IDEF14 method, developed the processes involved in a network design effort, and provided insights into the languages that are required to support the IDEF14 procedures.

Potential Areas for Future Work

A number of promising areas for additional work were identified during the development of the IDEF14 method. Several of these are listed below, along with a brief description of the benefits to be gained by pursuing further development along these lines.

1. Method refinement. A number of areas within the prototype method merit further development and testing. Among these are expansions to the techniques supporting the IDEF14 method procedure, further development of graphical language elements, the development of a computational language of expression (elaboration language), and so forth.

2. Research on the specific Queuing, Reliability, and Cost models that would best reflect the figures of merit for a design. Specification of the attributes of network components required to generate the models.

3. Mechanisms for generating the Queuing, Reliability, and Cost models from the network design.

4. Mechanisms to perform the queuing, reliability, and cost analyses on the models that are generated from a network design. Although the actual analyses are beyond the scope of the IDEF14 method, it is useful to identify algorithms, techniques, and best practices for carrying out such analyses.

5. Details on network design rational capture. The design rationale capture of the IDEF4 method and the IDEF6 design rational capture method could be modified and tailored for the IDEF14 method.

6. Details on decision support to be provided by the IDEF14 method for selection of design for implementation.

7. Validation of the IDEF14 network design method. The method must be validated by using it under various scenarios (modeling an AS-IS network, upgrading an existing network, and designing a network from scratch). A wide range of test situations covering these scenarios is recommended to maximize the robustness of the method.

8. Tools to help a network designer use the IDEF14 method. The success of any method depends heavily on automated tools. This has always been true and is likely to continue to be so in the foreseeable future. Automated tools assist practitioners in the application of a method, and provide a rapid and reliable means for sharing, storing, and reusing information.

9. Libraries of network components. On-line libraries of network components made available through the information superhighway would provide network designers with a mechanism for updating the component pools with libraries that are updated with state-of-the-art technologies. New alternatives to a network design problem could be made possible through such libraries.

Conclusions

Effective network design requires a formal method that allows for the collection of data, the creation and evaluation of multiple designs, and the selection of the final design. The network design process includes the modeling of the AS-IS network. Capturing the current and future workloads of the TO-BE network is an important step in the network design process. For each alternative design, a thorough analysis must be performed, and design rationale must be documented. The method must be versatile to incorporate use of state-of-the-art and future technologies of computer and communication networks. The IDEF14 method was designed to assist in the network design process.

Considerable progress has been made toward developing the foundations of the IDEF14 Network Design Method. In its envisioned form, IDEF14 provides a systematic and reliable approach for network designers to capture the AS-IS network, characterize workloads, create multiple design alternatives and select the best design for implementation. The current developments provide the foundation for future endeavors to refine and develop the IDEF14 method and effectively leverage the products of IDEF14 application.

IDEF14 Bibliography

[Colb 87] Colbourn, J. (1987). *The Combinatorics of Network Reliability*. The International Series of Monographs in Computer Science. New York: Oxford University Press.

[Klein 75] Kleinrock, L. (1975). *Queuing Systems — Volume 1: Theory*. New York: John Wiley & Sons.

[Klein 76] Kleinrock, L. (1976). *Queuing Systems — Volume 2: Computer Applications*. New York: John Wiley & Sons.

[KBSI 94] Knowledge Based Systems Inc. (1994). *IDEF5 Method Report*. Wright-Patterson Air Force Base, OH: AL/HRGA.

[Mayer 87] Mayer, R. J., et al. (1987). *Knowledge-based integrated information systems development methodologies plan* (Vol. 2) (DTIC-A195851).

[Mayer 92] Mayer, R. J., et al. (1992). *The IDEF3 Process Description Capture Method Report*. Wright-Patterson Air Force Base, OH: AL/HRGA.

[Minoli 91] Minoli, D. (1991). *Telecommunications Technology Handbook*. Norwood, MA: Artech House.

[Minoli 93a] Minoli, D. (1993). *Broadband Network Analysis and Design*. Norwood, MA: Artech House.

[Minoli 93b] Minoli, D., & Keinath, R. (1993). *Distributed Multimedia Through Broadband Communication Services*. Norwood, MA: Artech House.

[Minoli 93c] Minoli, D. (1993). *1st, 2nd, & Next Generation LANs*. McGraw-Hill Series on Computer Communications. New York: McGraw Hill, Inc.

[Newnan 80] Newnan, D. G. (1980). *Engineering Economics Analysis*. San Jose, CA: Engineering Press, Inc.

[Riggs 82] Riggs, J. L. (1982). *Engineering Economics*. New York: McGraw-Hill, Inc.

[Schwartz 88] Schwartz, M. (1988). *Telecommunication Networks: Protocols, Modeling and Analysis*. Addison-Wesley Publishing Company.

[Zachman 87] Zachman, J. (1987). A framework for information systems architecture, IBM Systems Journal, 26(3), 276-292.

TOWARD A HUMAN-SYSTEM INTERACTION DESIGN METHOD (IDEF8)

Introduction

This section provides an overview for the IDEF8 Human-System Interaction Design Method. IDEF8 is a method for producing high-quality designs of the interactions that occur between users and the systems they operate. Systems are characterized as a collection of objects which perform functions to accomplish a particular goal. The system with which the user interacts can be any system, not necessarily a computer program.

Human-system interactions are designed at three levels of specification within the IDEF8 method. The first level defines the philosophy of system operation and produces a set of models and textual descriptions of overall system processes. The second level of design specifies role-centered scenarios of system use. The third level of IDEF8 design is for human-system design detailing. At this level of design, IDEF8 provides a library of *metaphors* to help users and designers specify the desired behavior in terms of other objects whose behavior is more familiar. Metaphors provide a model of abstract concepts in terms of familiar, concrete objects and experiences. For example, a *light switch metaphor* might be used to specify interactions involving two possible options. Among the products of this level of design is a human-system interaction mock up with which to test user requirements, formulate user interface strategies (e.g., selecting preferred input and feedback devices), and so forth. Once validated, the products of IDEF8 application are used by system developers (e.g., programmers) to build implementations.

Much of IDEF8's language constructs are borrowed directly from the IDEF3 Process Description Capture method because IDEF8 needs a mechanism to capture and organize process information at multiple levels of abstraction and detail. Specialized language extensions distinguish IDEF8 design models, which are *prescriptive* in nature, from IDEF3 *descriptive* representations.

This section covers the basic concepts and elements of the IDEF8 method. The fundamental goals of IDEF8 come from the need to promote good design practice for human-in-the-loop systems to realize higher quality implementations in less time and at a reasonable cost. IDEF8 seeks to help users produce good human-system interaction designs and consequently higher quality systems by facilitating user-focused data collection, enabling direct user involvement in design activities, focusing efforts on early validation of designs using mock-ups and prototypes, and promoting more productive iterations through the design process.

What is Human-System Interaction Design?

User effect on the behavior of a system and system response to requests is the human-system interaction element of any system (e.g., a software/hardware system). Human-System Interaction Design (HSID) is the process of creating systems exhibiting interaction behavior that makes the system effective, easy to use, and more comfortable for the user. Often, these qualities are conflicting and hard to define. Most of us have experienced systems that have poor interaction characteristics (e.g., poor responses, less than intuitive user interfaces). Even systems that regarded as having very good user interface qualities have blemishes or illogical actions the user community would like corrected. While it is unlikely that any system will meet everyone's expectations, users generally share a common sense of what constitutes a 'good' or 'bad' interaction design. Explicit methods for HSID promote the development of systems that exhibit the 'good' qualities that users prefer. HSID tries to use proven techniques to develop designs quickly and effectively that will produce systems exhibiting desirable qualities.

The human-system interaction component captures how a human will command the system and how the system will respond. Design decisions directly affect the people who use the final system. An individual's emotions and mental perceptions may be positively or negatively affected by the way the system interacts with the user. The effect can be wide reaching in that organizational behavior (i.e., corporate culture) may need to adapt to new systems that do not conform.. Analysts study people in order to get the context and content right

during analysis. Designers need to study people to design the interaction specifics, using the interaction technologies available for a particular system.

What is IDEF8?

The IDEF8 method supports the modeling and design of interactions between human operators and the systems they use. Current human-system interaction development tools concentrate largely on the appearance of an interface. IDEF8 concentrates on specifying desired interaction behavior at three levels of design: the concept of operations level, at a role-specific level detailing scenarios of use, and at a detailed design level.

IDEF8 method is not a Graphical User Interface (GUI) development method. It can be used in conjunction with GUI development systems, but it is not strictly for GUI development. The method is not used to define the actual interface, but it is used to describe the interactions between the user and the system. The user knows what the tasks or actions should be, and IDEF8 is used to capture this information. Further, IDEF8 is not an interface standard like X Windows. It is designed to be used at a more abstract level. The method can be employed independently of the implementation choice for the GUI. While an IDEF8 model could specify how the interface should interact with the user, the method does not require that screen layouts, interface object types, or other specific interface information be collected. The main purpose of IDEF8 is to define interactions without defining interface details. Later, the interface will be built, user acceptance testing will be performed, and the results will be used to fine tune the interface. Thus, the IDEF8 method is used to design interactions between the system and users of the system at multiple levels of abstraction; and, the results of IDEF8 application are used in implementation and documentation situations. The following list summarizes the kinds of situations in which the IDEF8 products will be created and used.

1. Analyzing—The IDEF8 models are used as a way of understanding and modeling the interactions between humans and systems. The interactions with the system are captured in the IDEF8 models. Modeling current systems helps to identify weaknesses in the design or implementation.

2. Designing—IDEF8 may be used to design interactions between users and the system under development at multiple levels of abstraction.

3. Implementing—The IDEF8 models can be used to provide specifications to system implementors.

4. Documenting—The IDEF8 method may be used to document an existing system or to describe the design of a new system.

Depending on the needs of the project, the IDEF8 method may be used to support one or more stages of the development process.

Motivation for a Human-System Interaction Design Method

While there is a vast body of research in the cognitive sciences and human factors arenas for developing good user interfaces and a host of application building tools with Graphical User Interface (GUI) ‘parts’ or ‘components,’ little assistance has been provided to the developer in defining the context for user interface design. That is, human-system interaction designs often remain implicit, forcing users to rely on system developers to produce user interface strategies consistent with an implied concept of operations, role-specific use case scenarios, and interaction concepts. Applied use of knowledge and technology for developing computer interface designs follows the definition of what interaction is to be supported. It also follows the definition of how the interaction should be supported. Only after these elements have been defined can system developers effectively leverage application building tools to put the system together in a way that ‘looks’ and ‘feels’ right. The IDEF8 method formalizes this process to speed overall development time with better quality results and fewer iterations.

Rapid changes in technology prompt the need for a method to explicitly document intended human-system interaction. New interface paradigms and supporting technology are rapidly coming into common use. With this new technology comes the need to easily integrate these devices into existing and emerging systems. Using

IDEF8, logical user interactions with the system can be documented and organized while leaving implementation decisions (e.g., which input/output device) to the developer. IDEF8 provides an extensible set of interaction metaphors with supporting implementation templates to support such 'plug-and-play' development strategies.

There is a growing need to develop systems that can operate on multiple platforms while still exhibiting the same 'look and feel.' While actual implementation on different platforms will require platform-specific adjustments, the definition of the user interactions and system responses can be modeled in a general way using IDEF8. IDEF8 captures the intended interactions between the user and the system. The manner in which these interactions manifest themselves in the system can be left to the implementor or they may be completely specified. By providing designs at the human system interaction level, the IDEF8 method produces models of the system that can be reused and directly ported to other platforms.

Perhaps most importantly, users often find it difficult to articulate exactly how they want a system to operate. Yet, generally, users are able to describe how they envision the general concept of operation and isolated elements of the system in operation. In providing these descriptions, users often reference features or operational characteristics of other objects or systems. That is, users often express how they want the system to behave or interact by comparing desired behavior with the behavior of objects that are mutually familiar. For example, a user may specify the desire to control air temperature for an automobile cabin heating and cooling system using a single control knob that operates like a volume control knob on the radio. This metaphor would indicate that designs using multi-position switches (offering a far more limited range of temperature options) and separate controls for heating and air conditioning could be unacceptable. IDEF8 leverages this tendency to specify design goals in terms of metaphors to help articulate and narrow the design space. While accomplishing this goal, IDEF8 helps development teams resolve contending objectives, goals, priorities, and measures of success by promoting effective multidisciplinary activity with mechanisms to support communication among development team members.

IDEF8 Products

Application of the IDEF8 method will produce:

1. Descriptions of "AS-IS" Human-System interactions. IDEF8 can be used to collect and model current system interactions with the user, including both normal and abnormal interaction situations.

2. An implementation-independent definition of the "TO-BE" design for a system enabling cross platform development. IDEF8 captures multiple levels of design specification including human-system interaction at a concept of system operation level, a role-centered scenario of use level, and at the detailed gesture-response level.

3. A mock-up of the designed human-system interaction and a user-validated human-system design. IDEF8 provides a robust library of implementation templates for selected design metaphors and a discipline for involving the user in validating human-system interaction design correctness.

Although the primary use of IDEF8 is to define the "TO-BE" requirements by identifying the human-system interactions that should be supported in the implementation of the system, the use of IDEF8 for "AS-IS" models is also important.

Users and Key Beneficiaries of IDEF8

Two broad categories of users for an IDEF8 method can be considered. The first category of users are those who would directly apply the method to design human-system interaction. This class of users ranges from those who manage HSID efforts to those who perform the detailed implementation of target systems. The individuals targeted as direct users of IDEF8 include managers and practitioners of enterprise improvement initiatives (e.g., Total Quality Management, Business Reengineering), strategic systems planners, systems developers, and end users. The second category of users are those individuals who use the products of an IDEF8 application effort. Business owners and managers are among this set of users. For them, the most visible

element of an IDEF8 method is not the set of techniques used to define system operational scenarios or metaphors of human-system interaction, but the system operational scenarios themselves. System implementors are also among this class of users. These individuals use the products of IDEF8 to guide user interface design and overall system implementation activities.

Potential Benefits

This section describes some benefits of the IDEF8 method.

Higher quality systems produced in less time at lower cost. User interface development requires a major portion of the software system development budget [Boies 89]. This along with the fact that the interface is a determining factor in market appeal makes lowering the cost a primary concern. Explicit definition of human-system interaction helps avoid costly iterations in user interface design while ensuring correct operation; i.e., operation that is consistent with the intended scenarios of use. Even small changes to the user interface can cause major problems with budget and schedules. The challenge is to identify weaknesses in design early, to prevent this situation. The IDEF8 method, by formalizing and organizing the HSID process, helps to reduce time and errors, thus lowering the cost.

User-friendly user involvement in the design process. The more ill-defined a system's concept of operation, scenarios of use, and 'look and feel' are, the more likely the system will not meet user quality expectations. Traditional methods focus largely on interface design accomplished in an ad hoc fashion without an explicit definition or design for human-system interaction. User acceptance of the resulting interface is left to chance, relying on the experience and knowledge of system developers of the application domain. The IDEF8 method provides an explicit mechanism to involve users productively in the earliest stages of the design process in participative, team-based design activity. IDEF8 also provides a mechanism for working directly with users to refine human-system interaction designs incrementally. This leads to the development of actual implementations without requiring an understanding of potentially applicable implementation technologies.

Methodization of Human-System Interaction Design practices. Most designers find that HSID is an ad hoc process at best. It is difficult for the various designers of different disciplines (i.e., engineers, programmers, psychologists, artists, etc.) to prioritize factors that are the most important to the success of the system. HSID is complex. Having a formalized method for handling this complexity replaces the ad hoc processes with a more concrete and manageable process.

Common language for cross-platform design, system documentation, and interface description. Currently, there are not any common languages for describing interactions at the abstract level. The IDEF8 method provides a means to describe interactions in generic terms for use in documenting and designing cross platform systems.

Systems that look, feel, and operate as intended. What does the user want to do? This is the central problem that the system should solve. It is odd that systems often are cluttered with features, but unable to provide the user with a simple means of performing tasks. IDEF8 focuses on the tasks that are to be performed and the interaction that is required.

Provide a model for analysis. By documenting interaction designs using IDEF8, many problems can be identified that otherwise would not be easily found. The IDEF8 method forces system designers to examine desired interactions closely, and thus uncover problems that may only be found later by users.

Enforce a formulation and commitment to an interaction philosophy. The use of IDEF8 in the design forces the designers to carefully select and commit to an interaction philosophy. This lends a consistency and "look and feel" to the system that probably would not occur otherwise.

Summary of Developments and Research Findings

IDEF8 is still in the developmental stage and does not possess stringent rules and procedures or well defined syntax and semantics. But, as it draws on other IDEF methods for basic techniques and graphical

languages, it is still usable. The following section describes the current state of the research and the preliminary ideas about IDEF8.

In the design of any method, there is a set of design goals that the method should attempt to meet. While many compromises and tradeoffs must be made during the evolution of the method, the design goals and principles are always the primary objectives of the method design. In the design of IDEF8, these principles were identified as design objectives. They represent ideals that the method should promote to aid users in the development of good models of the system. The method should also promote the creation of models that result in user friendly human-system interaction. That is, the models must do more than just capture the interaction between systems and users; they must also help promote sound user friendly interactions in the design of systems. The main theme that runs through these principles and goals is reducing the cognitive load on the end user. In the past, HSID has been largely an ad hoc discipline;, this can be changed by paying close attention to vital HSID concepts. The following lists describe the design goals and principles embodied in an IDEF8 systems development strategy.

IDEF8 Method Design Goals

The design goals and principles that are important to human-system interaction design are listed below. The IDEF8 procedure includes a step in which the design team must identify the pertinent principles for the design of the system and then prioritize and develop success criteria for meeting them. The list is not all encompassing, all items are not equal, and every item is not necessary for every system. Each system and each user community is different, and the design goals and principles should reflect these differences.

Compatibility—Minimize the amount of information processing that will be necessary in general with human perception, memory, problem solving, action, communication, and in particular with the class of users that will be using the system.

Consistency—Minimize the difference in dialogue within and across various human-system interfaces.

Memory—Minimize the amount of information that the user must maintain in short-term memory. The goal is to reduce the amount of information needed in short term memory, especially, if other information processing tasks are required simultaneously.

Structure—Assist the users in developing a conceptual representation of the structure of the system so that they can navigate through the interface.

Feedback—Provide the user with feedback and error-correction capabilities.

History—Provide users with a history of past user actions possibly allowing them to ‘redo’ the same or similar commands.

Guidance—Guide users through the task or activity to completion.

Identification—Provide the user with a way to identify interface objects.

Transition—Provide users with a mechanism to orient themselves within the system (e.g., one which helps to establish where you are and where you’ve been).

Choice—Provide a clear and complete set of actions that can be selected by the user.

Demonstration—If users do not know what is possible, they should have a way of getting assistance.(What is possible with what? Who will assist? Unclear)

Explanation—On-line help systems are being viewed more and more as a requirement rather than a option.

Workload—Keep user mental workload within acceptable limits.

Individualization—Accommodate individual differences among users through automatic adaptation or by user-tailoring the interface.

Direct manipulation—Favor object-like paradigms.

See-and-point—As opposed to remember-and-type.

Stress What You See Is What You Get.

Forgiveness—Provide undo and cancel functions; minimize irrevocable user actions.

Constraints—Allow only correct choices instead of backing out from error state.

Aesthetic integrity—Ensure a consistent "look and feel."

User control—Allow for user preference and changeable options.

Transparency—Provide intuitive user control; the interface should not get in the way of the user's tasks.

The objective is to optimize human-system interaction in terms of these design goals and principles. These objectives helped to guide IDEF8 method developments.

The previous list illustrates some of the qualities that are desirable in the target system. The following list illustrates some of the qualities required of the IDEF8 method to achieve these goals.

Early focus on users—Direct contact with potential users prior to overall system design. The user interface should be the first component developed in the system design.

Interactive design—Inclusion of typical users into the design team, at least at the beginning of the design.

Empirical measurement—Evaluation of the learnability and usability of the interface as well as conducting empirical and experimental studies throughout the development process.

Iterative design—Incorporation of the results of behavioral testing into the next version of the system.

Reduction of design time and design errors—Gives the designers and developers a consistent language to use during design process.

Design documentation—Provides traceability and consistency checking for the developers.

Focus design process on user requirements—Reduces the chance of ad hoc user interface development.

Provide the user with a model of the system—Resulting models can be used for system documentation and user help systems.

Use metaphors from the real world—Such as buttons, menus, etc.

These lists are not meant to be complete, and one can see that many of the goals are conflicting and interdependent. The best thing to do is to strike a balance that best suits the current application and perform user acceptance testing to fine tune the human system interactions. The challenge is to create a well behaved

interface that helps the users access and complete their required tasks without getting in the way. The best interfaces do not appear to be interfaces at all.

Basic Concepts in IDEF8

This section describes some of the basic concepts for the method.

Process

A process is an ordered sequence of events. In human-designed systems, the events that constitute a process are designed and ordered to achieve some desired outcome.

Input/Output Item (IOI)

An Input/Output Item is any item which conveys information from the user or to the user. IOIs include, but are not limited to, graphics, text, icons, sound, and tactile feedback. Future versions of IDEF8 should include an IOI dictionary, which is a repository of the information known about a piece of information. This dictionary will include the information about the IOI, for example: (1) Name, (2) Type, (3) Display Method (defaulted by type), (4) Color, (5) Font(s), (6) Icons, (7) Graphics, and (8) Description. The dictionary will also include information about the types or kinds of input items.

Metaphor

Metaphors are models of abstract concepts given in terms of familiar, concrete objects and experiences. This is true if we are using metaphors from our language or metaphors from an interface. Because metaphors are used as models, users have difficulty when a metaphor suggests an incorrect model. Some metaphors are like real world objects (e.g., buttons), while some are unlike the real world but provide a usable model for users (e.g., grabbing a drawing object on the screen is not like grabbing a real world object). IDEF8 includes metaphors in the method and models them as templates to be reused during the HSID.

The tables in this section are initial attempts to identify possible objects, operations, gestures, and devices that will occur in the design and use of metaphors. The tables are not complete—new devices and new metaphors are created as new technology becomes available.

Table 6. User Gestures and Interactions Table

Objects	Operations	Gestures	Primitive Gestures
Button	Select	Drag & Drop	Point, Click, Hold, Point, Release
Form	Choose	Type	Type
Window	Stretch	Point	Move, Stop
Screen	Shrink	Drag	Point, Click, Hold, Move
Text Field	Rotate	Grab	Point, Click, Hold
Menu	Zoom	Grab & Pan	Point, Click, Hold, Move
Selection List	Pan	Click	Press, Release
Combo Box	Scroll	Press	Press
Grip	Move	Hold	Hold
Handle	Create	Release	Release
Thumb button	Open	Stop	Stop
Scrollbar	Close	Move	Move
Check Box	Save		
Various Icons	Edit		
Desktop	Insert		
Pointer	Delete		
Cursor	Clear		
Spinner	Find		
Scroll Text	Replace		
Radio Buttons	Undo		
	Repeat		
	Cut		
	Copy		
	Paste		
	Toggle		
	Help		
	Describe		
	Abort		
	Redo		

The following is an attempt to categorize the various ways the user can interact with the system, both physically and logically. For example, users may want to specify either a physical button on an actual device (e.g., a mouse) or a logical button (e.g., an OK button on the screen).

Table 7. I/O Devices

Physical Input Devices	Physical Output Devices	Logical Input
Keyboard	Printer	Text Entry
3-D Mouse	CRT	Movement
Track Ball	Light	Keypress
Light Pen	Gauge	Buttonpress
Joystick	Dial	
Glove	Speaker	
Digitizer	Virtual Reality Glasses	
Graphics Tablet	Pressure Glove	
Button	Head Up/Down Displays	
Dial		
Knob		
Touch Screen		
Touchpad		
Microphone		

To support the metaphor design goal, IDEF8 contains a catalog and library of metaphor templates that can be used in the interaction models. These templates are derived from the various metaphors and devices given in the tables.

The use of metaphors is a natural way to convey the model on which the interface operates. Figure 29 illustrates some of the common metaphors used in many systems. The decision to choose one metaphor over another depends on what the behavior of the system should be.

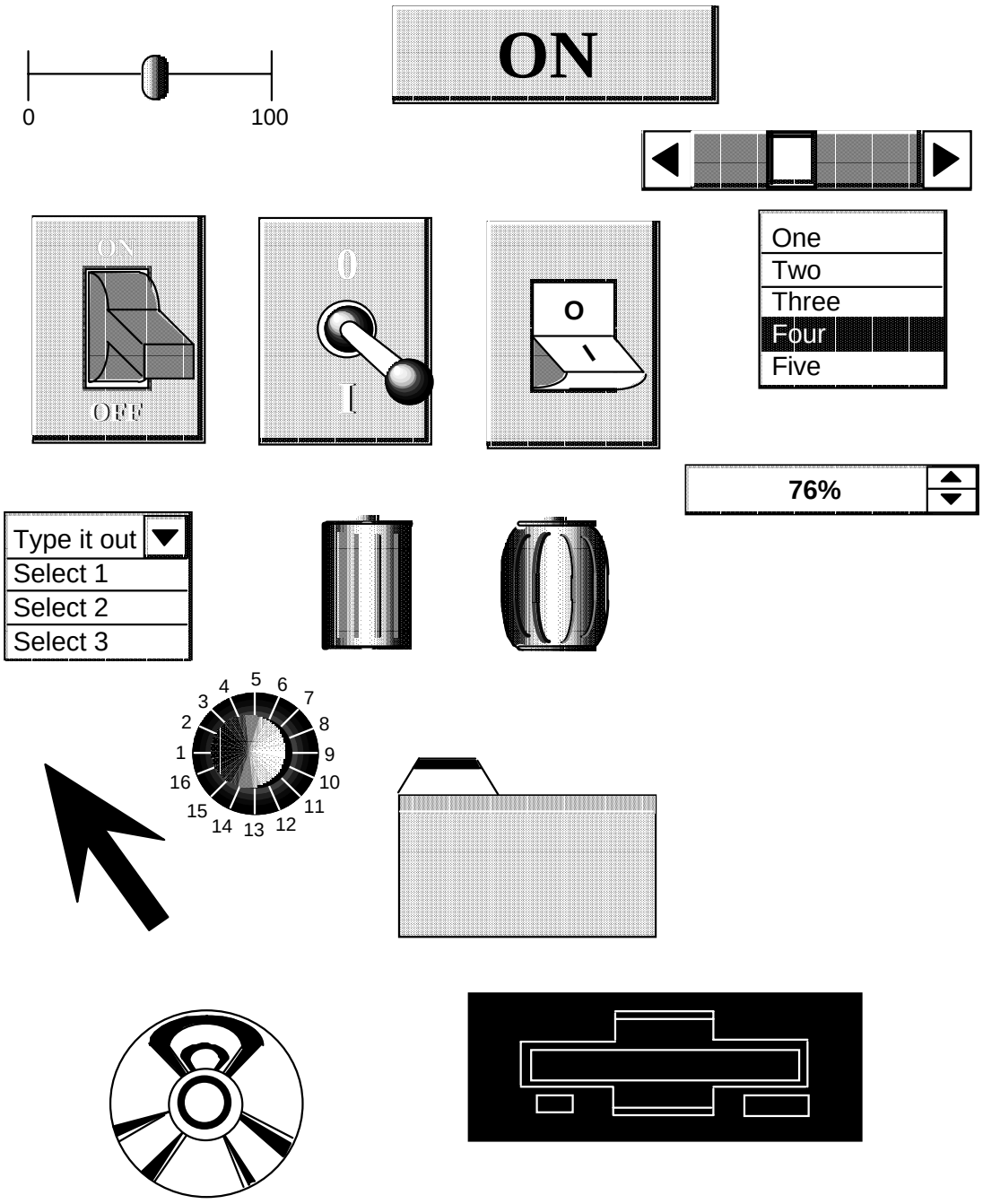


Figure 29.
Common Metaphors

For example, suppose the situation requires the user to respond to a yes or no query. There are various interaction metaphors that could be used such as a toggle switch, a check box, a list box with 'yes' or 'no' listed, or even a text entry field requiring the user to type 'y','e','s' or 'n','o' and press return. The same user interaction scenario can be effected by different metaphorical constructs. All of the examples will support the same user interaction, however some are more appropriate than others.

IDEF8 uses metaphors to help select the best model of interaction and then instantiate a template that represents the metaphor in detail. The blank template is filled in with the specific requirements of the particular user interaction and added to the detailed HSID model. To illustrate, take the following figures showing an interrupt message box with a message, and two buttons for the two options that the user may take. The system

view is described by two process boxes, display dialog and perform actions. These processes are shown to decompose into more detailed processes. The user view is described from the user perspective and a similar structure is shown in the Figure 31.

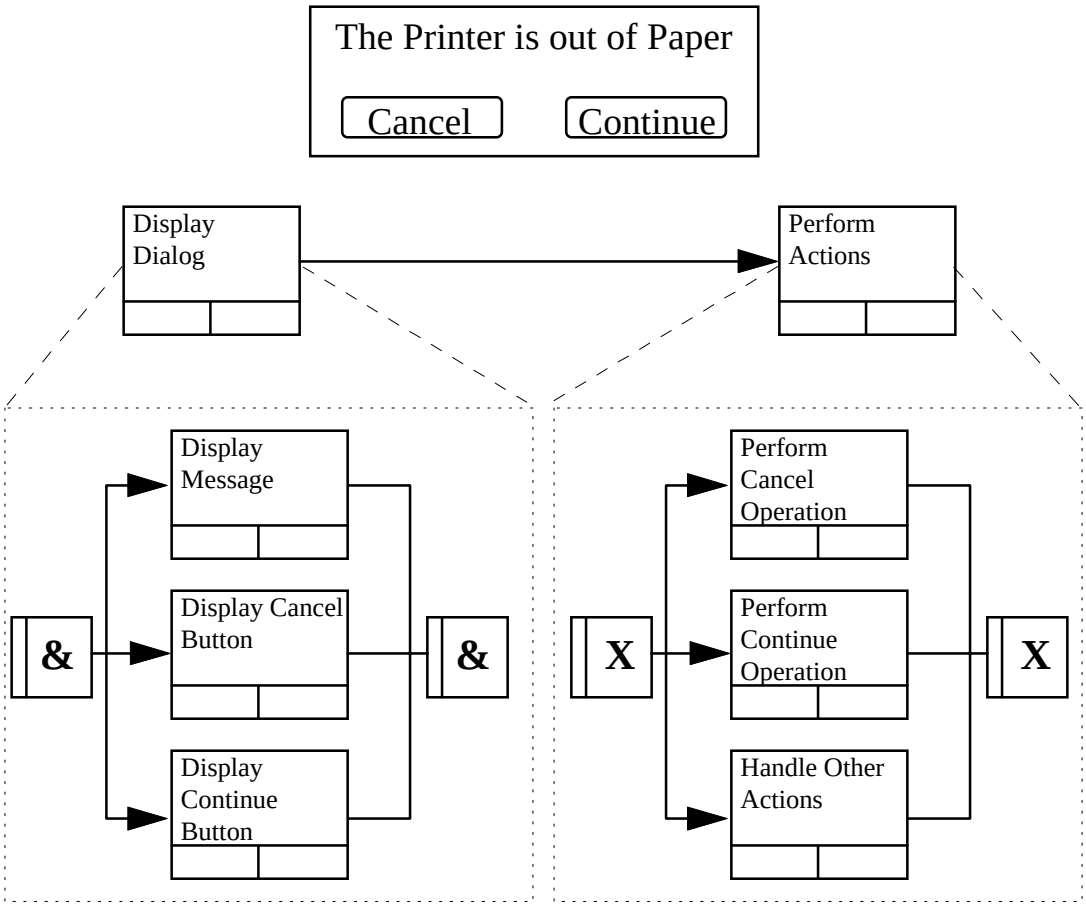


Figure 30.
System View of Out of Paper Dialog Message

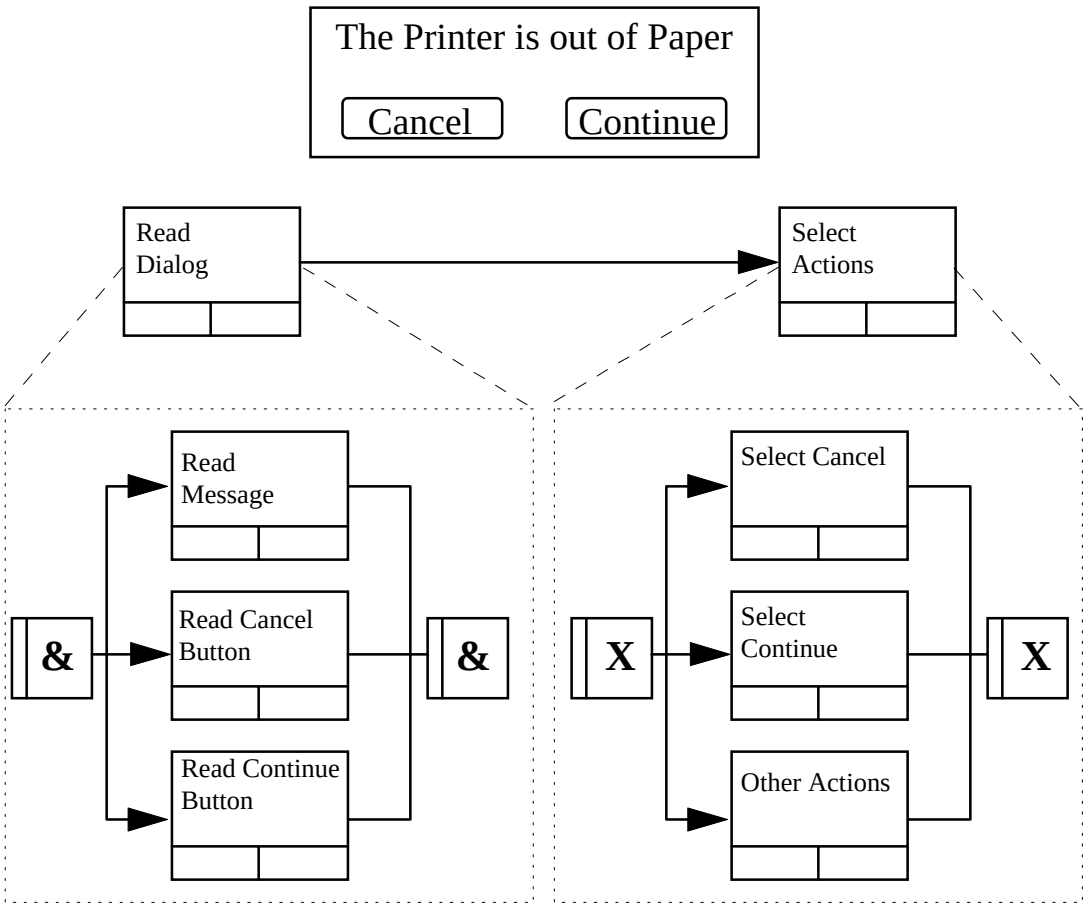


Figure 31.
User View of Out of Paper Dialog Message

To see how the interactions are modeled, Figure 32 contains an interaction diagram that shows how these processes are instantiated over time. The user/system dichotomy is shown in this diagram. The information is conveyed across the interface by the precedence arrow showing the ordering of the processes.

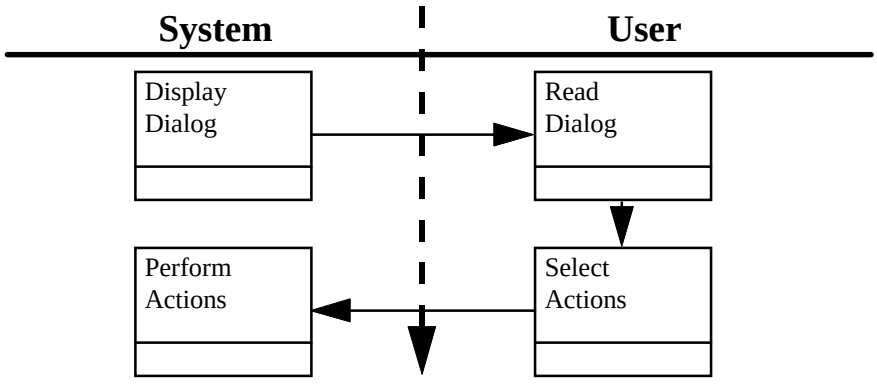


Figure 32.
Interaction Diagram of Out of Paper Dialog Message

A more detailed example involves the resize box scenario. Anyone who has used a drawing program has had the opportunity to draw a rectangle and resize it. This example breaks down the metaphor into component steps and describes the user and system interactions involved.

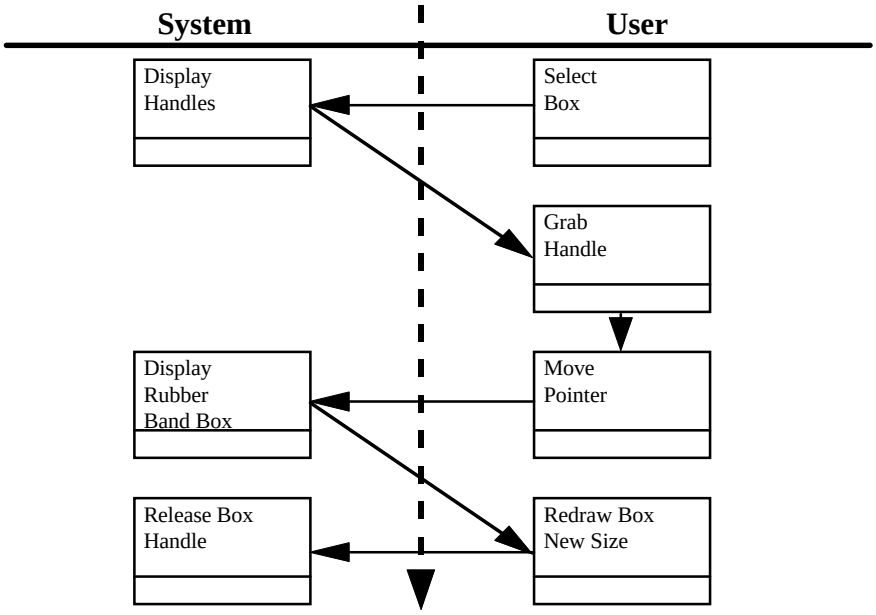


Figure 33.
Interaction Diagram of Resize Box Example

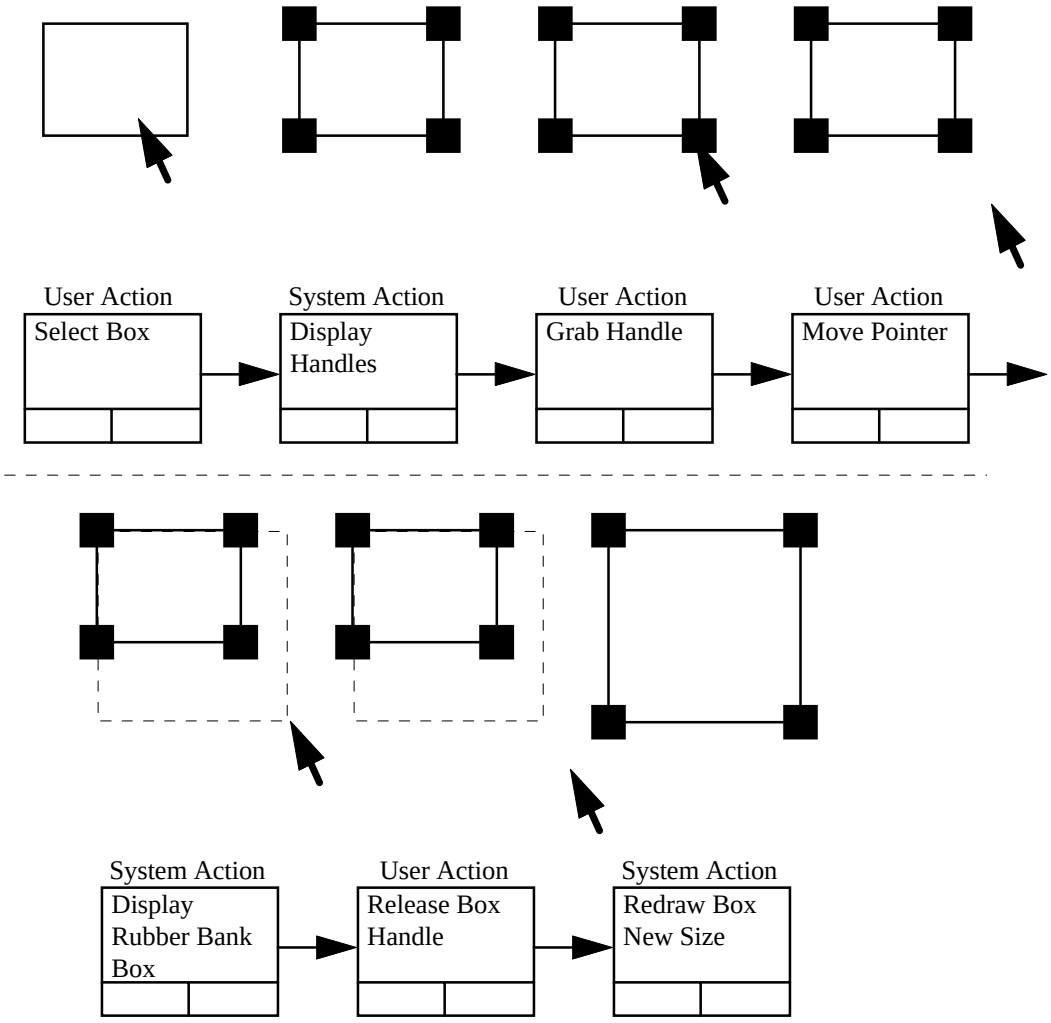


Figure 34.
Resize Box Example Process by Process Comparison to Screen

Interaction Templates

As mentioned previously, IDEF8 has templates of the various metaphors that can be incorporated into an interaction description. There are templates for buttons, lists, combo boxes and so forth. The concept is to have a standard template for each metaphor that can be filled with information peculiar to the system being designed. The IDEF3 process description capture language appears to be suitable as a mechanism to represent these templates, although some tailoring may prove helpful. For example, experimentation revealed that it may be useful to provide some way of marking activities that are accomplished by the user and activities that are accomplished by the system.

Template Catalog and Library

The template catalog and library contains the various templates of metaphors with an annotated catalog of descriptions that may be reused by the modeler. They are used to help the modeler find the kinds of metaphors that most closely characterize the desired interaction behavior. As the library grows to include new metaphors, the catalog is updated and annotated to characterize how the new metaphor templates are best applied. How this catalog will be structured and implemented has not yet been fully developed.

IDEF8 Procedure Developments

This section presents a prototype procedure for human-system interaction design. The procedure assumes a team approach with the primary roles in design being filled by targeted end users and designers. The following list describes the various user roles that may be involved in an IDEF8 project.

1. Managers are ultimately responsible for the project in which the IDEF8 method is used. These members of the development team are involved in the administrative activities of the system development. The manager is responsible for providing project and technical systems direction, ensuring project coordination, and managing each process of the system development activities. They are responsible for all of the formal project documentation, ensuring that materials are completed and comply with standards. The manager advises the customer on decisions about design, coordinates review cycles and related problems, and interacts with the customer on project direction type questions. Strategic decision making is the main function of the manager.

2. Modelers are responsible for generating the IDEF8 model of human-system interaction. The modeler retrieves the necessary information from the user. Modelers are responsible for the technical activities of system development. The modeler is an expert in the use of the IDEF8 method and is responsible for generating the necessary models and documentation required for the success of the system development. Close contact with the customer and the end users is necessary.

3. End Users will use the system that is being modeled. The system may be one in use at the present time ("AS-IS" model) or it may be a system that is being developed ("TO-BE" model). In either case, the user is the one whose interactions are described as they pertain to the system. It is the end users who will benefit from the development effort. The end users are the source from which most of the data is collected for the IDEF8 models. They are responsible for providing feedback regarding the accuracy of the models that are created.

4. Reviewers are responsible for examining the model generated by the Modeler. This function is designed to help the Modeler maintain consistency between the system and the model of the system. The reviewer is responsible for validating the models against the existing or proposed system.

5. Designers are responsible for identifying and evaluating solution alternatives for the design of the system. They are members of the development team who create a plan for implementing the system. They must analyze the system and its environment, develop solutions, evaluate alternatives, and select the appropriate design for implementation.

6. Implementors are responsible for taking the IDEF8 models and creating the system and its interface. These members of the design team are involved in the physical realization of the system from abstract descriptions and models. They must prepare the design for implementation, fabricate the various components, assemble the system components, and ensure the quality of the components.

IDEF8 HSID is an interactive process with three primary modes of activity, each representing more detailed levels of design. These modes include:

Mode 1 - Define Philosophy of System Operation

Mode 2 - Design Scenarios of Use

Mode 3 - Detail Human-System Interaction Design

Iteration between modes is encouraged to increase usability and user satisfaction with the system. The procedure is not necessarily sequential. IDEF8 is meant to be used in parallel with other design activities (e.g., database design). Other activities can occur in parallel with IDEF8 application. IDEF8 is designed to support and draw from related design activities. Thus, to apply IDEF8 at the expense of other design methods is not advisable.

The first mode sets the scope of the system and specifies the design objectives which include making strategic decisions and determining success criteria. Critical system functions are identified, high-level allocations are made and overriding system constraints are identified. The product of this mode is an explicit definition of the critical system functions and processes characterizing the system's Concept of Operations. A prioritized list of critical functions and constraints is also produced. This mode of design, and those that follow, is accompanied by iterative kit reviews and structured demonstrations to provide a validated baseline for Mode Two design activities.

The second mode of IDEF8 design centers on role-specific scenarios of use. This mode begins by identifying and classifying the various user roles involved in the system. Once the roles have been specified, role-specific scenarios of use can be described using specialized conventions for an IDEF3-based HSID language. Next, task/function analysis is performed. Detailed examinations are performed to ensure that the required attributes and services—the required content—are specified. Required modifications revealed through kit reviews and structured demonstrations are also accomplished. The products of the second mode are models of the various scenarios and the results of the task/function analysis.

Mode three begins with more detailed requirements and specifications collected in the dialog guideline definition process. Based on the guidelines, metaphors are selected and models of detailed human-system interaction are created. As with the other modes, mode three includes kit reviews and demonstrations. When a consensus is reached, human-system interaction mock-ups are constructed to evaluate how well the tradeoffs and compromises work in a real world setting. Mock-up development and testing are used to validate design concepts and to elicit additional requirements. Details of actual interaction are specified during this mode. This may include designing the format of windows and reports. Prototyping is used to help develop and select the interaction mechanism. This activity is distinguished from mock-up development by constructing working pieces of the designed system whereas mock-ups merely support mental validation of design specifications. Finally, users test and evaluate the prototype. Prototype test results are used to iteratively improve the design.

Mode One: Define Philosophy of System Operation

A philosophy of system operation must be defined to establish the context for team design activity. High-level IDEF3 process descriptions, IDEFØ function models, and structured text are used together to define the philosophy of operation. These may be packaged in a concept of operations document as one of the key products of the Mode One activity.

Scope System

Defining the scope of the system involves specifying what functions and processes are included in the system and which are explicitly excluded.

Specify Design Objectives

The IDEF8 procedure is an iterative process wherein the first step is to specify the criteria to measure progress and determine when it is time to stop. The design objectives determined during this step should be reviewed periodically throughout the IDEF8 project to determine whether project cost, quality, and schedule goals are being met.

Determining design criteria for evaluation requires the method user to specify measurable usability objectives. Objectives define the usability to which the system must adhere. These objectives drive the iterative design and help to manage the design. They also focus on the users and what the users will be doing.

Objectives should include user input (measured subjective criteria about how users liked the prototype), and user performance measures (objective observations about how well users performed using the interface).

Before the design of the interface can start, usability objectives must be defined. There are four steps to setting these objectives: identify users and tasks, choose and prioritize design goals and principles, define usability criteria, and set levels of success for criteria.

Identify users and tasks. User identification, classification, and task identification are best accomplished by interviewing and observing users who will support or be supported by the system. Classification of the users identified through this step is often useful. Users may be classified by:

1. Skill level (e.g., Novice, Occasional, Intermediate, Advanced)

2. Organizational level (e.g., Executive, Officer, Staff, Supervisor, Clerk)

3. Membership in different groups (e.g., staff, customer)

For each category of user defined in the previous step, we must consider and tabulate the following: Who, Purpose, Characteristics (age, education level , limitations, etc.), and Critical success factors (needs or wants and likes, dislikes, or biases).

Choose and prioritize design goals and principles. It is necessary to define which design goals and principles are most relevant to effectively manage and measure usability for the system. Because usability involves a broad range of categories with respect to human-system interaction and human factors engineering, designers must select and prioritize those principles and design goals that are most important for the given system.

Define usability criteria. After the design goals and principles have been selected, the next step is to define the criteria for measuring the system's success. Metrics must be defined and should include the context in which they will be collected. Measures may be user opinion such as satisfaction ratings or user performance such as task completion time and number of errors.

Set levels of success for design goals and principles. Each of the usability criteria should have three attainment levels: minimum level, planned level, and best case level. Each criteria is stated in terms of previously defined metrics for each of the three levels. The minimum level is the cut-off between success and failure. Any measurements below minimum should be corrected through repeated iterations of the design. The planned level is the goal that is currently being sought for this iteration of the process. Lastly, the best case level represents a level that might be reached with more effort and/or resources.

Identify Critical Functions

The key functions to be supported by the system are listed previously as part of the scope definition activity. These should be prioritized in terms of their criticality by surveying the targeted users of the system. A critical functions list will be produced from this step to guide design activity.

High Level Allocation of Critical Functions

At the most abstract level, critical functions will be allocated to human operators or to the system. This activity attempts to balance the strengths of the human and technology with economic, cultural, organization, cost, and risk factors.

Identify Constraints

Constraint identification and management is central to all design activity. At this stage in design, constraint identification is largely isolated to constraints on possible design alternatives rather than among them.

Perform Kit Reviews and Structured Walk-Throughs

The goal of this step is to involve targeted users in a structured review approach. Two mechanisms for user review are provided in IDEF8: kits and walk-throughs. Both are patterned after the kit review and walk-through processes of previous IDEF methods. Users examine the IDEF8 models and identify voids, problems, and potentially useful modifications that are used to revise design artifacts.

Mode Two: Design Scenarios of Use

Identify User Roles

Examine the intended users and identify the various user roles that will make up the user community for the system.

Design Role Specific Scenarios of Use

For this step, use the IDEF3 method with IDEF8 extensions to model the human-system interactions based on the roles that were determined in the previous step.

Task/Function Analysis

Specify Interface Input/Output Representations

Study the existing user interaction metaphors and guidelines, as well as any established interaction metaphors and guidelines. Then, specify the interface representation that meet the requirements for the system being designed.

Specify the Control Structure for Interfaces, Dialogs, and Computations

Establish an initial command hierarchy. These may be presented to the users as a series of menu screens, a menu bar, or a series of icons that take actions when something is dropped on them. A command hierarchy is a presentation of available services, organized using procedural abstraction. Next, refine the command hierarchy. To refine the hierarchy, consider ordering, whole-part chunking, breadth versus depth, and minimal steps.

Ordering: Select distinct Service names, and order the Service names according to frequency of use and in a customary work-step order.

Whole-Part Chunking: Group related Services together.

Breadth and depth chunking: Stay within human short-term memory limitations. One guideline is to limit the maximum number of items presented to the user to between 5 and 9. Another guideline is to limit breadth to 3 chunks (i.e., pieces of information) of 3 and depth to 3 levels.

Minimal Steps: Minimize the number of clicks, drags, and key combinations to get the job done and provide shortcuts for advanced users.

Perform Kit Reviews and Structured Walk-Throughs

Once again, as in mode one, iteratively review Mode Two designs with the team members, reach consensus, and incorporate necessary changes into the design.

Mode Three: Detail Human-System Interaction Design

Define Dialog Guidelines

Determine and resolve tradeoffs. This is one of the more difficult steps. Use the design goals and principles to determine compromises for the system. Future iterations may change these decisions as new situations require.

Identify Failure Modes/Exceptions/Interrupts. This step concentrates on finding the abnormal interactions between human and system. Determine severity of each failure mode and specify what the response should be. Once again, repetitive iteration of this step will turn up new situations and old situation may disappear.

Create Input/Output Item Dictionary. The IOI Dictionary should contain the IOIs that are used in the model along with their descriptive information.

Design the human interaction component classes. Begin by organizing the human interaction design by windows and components.

Select Metaphors

User-centered scenario descriptions of desired human-system interactions establish the framework for selecting appropriate interaction metaphors. Direct user involvement in metaphor selection is critical to this task.

Perform Kit Reviews and Structured Walk-Throughs

As in modes one and two, review the design with the team members, reach consensus, and incorporate the necessary changes into the design.

Construct and Test HSID Mock-Up

In this task, role-specific system interaction process definitions and their accompanying metaphors, dialog guidelines, and templates are assembled to produce an integrated model of human-system interaction. This comprehensive model constitutes a system 'mock-up.' The mock-up is used primarily to validate system process designs and the general philosophy of human-system interaction. This task may be followed by, or merged with, prototype development.

Prototype and Test the System

Prototype User Interface. Prototyping the human-computer interaction is essential for the human-system interaction component. It should occur in both analysis and design. Human/system interaction design is more than look-and-feel. Observe the human as they use the prototype.

Use Interface Design Tools. Take maximum advantage of the many interface design tools available to get the prototype looking similar to the real system.

User Defined Interfaces. Use a facade to let the user design an interface to his or her satisfaction. By this step, the interactions have all been defined, so the information exchanged between the user and system is given in the IDEF8 models. Letting users design their own interface for some applications may be the best way to obtain the greatest user satisfaction.

User Acceptance Testing

It would be virtually impossible to design human system interactions without a rigorous set of experimentation and user acceptance testing. In this step, all previous design work will be tested. The metrics and measures defined in the first step will be used in the user acceptance testing phase. In addition to evaluating metrics against the design goals selected in the first step, the following metrics should be used to provide empirical information about user performance and the users' opinions of the system:

Time on Task. The amount of time it takes the user to successfully complete the given task using the system. There are three criteria for task completion: task is completed correctly, performed within specified time limit, and performed without intervention or direct instruction.

Completion Rate. The completion rate is the percentage of users that successfully complete the task being measured.

Error Free Rate. The error free rate is the percentage of users completing the task with zero errors. This is a stringent indicator.

Problem Classification. Problem classification provides information about the nature and severity of the problems and provides diagnostic information for future revisions of the system.

User Opinion. User opinion provides data about the usability of the system.

Iterate as Required

The activities comprising IDEF8's procedure should be considered "modes of thought" rather than sequential steps. Users should not expect to apply these activities in a strictly sequential manner. These modes of activity may be organized into phases to assist with management of the project. Iteration across modes may be required to satisfy established design goals.

Significant Accomplishments

A number of significant contributions of the IDEF8 method development are given in this section. Although IDEF8 is not a mature method, its initial development has produced several useful results.

1. *Identified HSID principles and design goals.* The preliminary work on IDEF8 identified the useful design principles and goals for the IDEF8 method and the target systems being modeled. These principles and goals were taken from various human factors sources. They help guide the development of the system so that it embodies the qualities that its users find pleasant and easy to use. These goals are not a complete list and some of them clash with one another which is why building the interfaces between humans and machines is so challenging.

2. *Developed innovative metaphor and gestures concept.* The IDEF8 method includes support for user friendly involvement in the definition of design concepts by introducing the concept of metaphors to characterize desired system interaction. The IDEF8 method uses metaphors to record and catalog the templates that can be used in the design of system interactions.

3. *Developed the concept of templates, catalogs, and template libraries.* Metaphor templates allow the modeler to develop IDEF8 interaction descriptions quickly by selecting appropriate blank templates from the library and filling in the specific data for the given interaction. Having these templates available greatly enhances the speed and accuracy with which the models are constructed. The catalog and library paradigm helps the modeler select the best metaphors to put into use in the interaction model.

4. *Developed a preliminary method procedure supporting iterative design.* A simplified procedure for HSID has been developed for the IDEF8 method. The procedure is also designed to promote maximized flexibility toward integrating IDEF8 application with other design methods.

Potential Areas for Future Work

IDEF8 is a promising method which supports HSID. However, there is still much work before IDEF8 reaches full maturity as an IDEF method. This section summarizes some of the areas where future work should be concentrated.

1. *Continue method development and refinement*. IDEF8 is still in a preliminary stage of development. There are more areas that must be investigated to further the development of the method. The current version a starting point but to achieve the kind of leverage that is envisioned, more development should be done.

2. *Incorporate storyboarding techniques.* Can a formalized storyboard paradigm be included in the method? Storyboarding is a powerful graphical description of a scenario. IDEF3 is a "box and arrow" version of storyboarding, but the actual storyboard technique may be more useful in certain cases.

3. *Develop/populate template library*. The template library is one of the keys to quick and efficient use of the IDEF3 method in IDEF8. Currently, only some of the templates have been created for some of the common metaphors. The library should be as comprehensive as possible, as well as open for extension as new metaphors are applied to interaction scenarios.

4. *Create better notation to support events, interrupts, and exception handling*. A great deal of user interface interaction is event driven. New notations for representing certain kinds of events and interrupts need to be investigated further. The use of IDEF3 in its current form is usable; however, better notation and extensions could be found to make it easier to represent interactions. What about what the system response back to the user? IDEF8 uses gestures and template metaphors to represent how to *do* things to the system, but the system conversely does things back (i.e. Clear Screen, Display Message). It may be necessary to have a list of prompts or actions by the system. Some of this behavior is encapsulated in templates that have been created so far, but more investigation may produce a more facile technique for this representation.

5. *Visual techniques for user views*. IDEF8 may need to describe what the user will see—not the exact layout, but a complete list of what is viewable. This can be inferred from the IDEF3 description, however it might be better to have a separate list.

IDEF8 Bibliography

[Boies & Henry 89] Boies, S. J., & Henry, S. C. (1989). Managing the Development of Effective User Interface. *Proceedings of the Thirteenth Annual International Computer Software and Applications Conference*. Orlando, Florida. 2.1–2.25.

[Bourbaki 92] Bourbaki, N. (1992). Building A Class Browser Using CLIM. *AI Expert*, 17–23.

[Brown 89] Brown, G. R. (1989). Control System Design:Part 4—The Man/Machine Interface:Designing Interactive Graphics. *Intech*, 34–39.

[Carroll 92] Carroll, J. M., Rosson, M. B., & Singley, M. K. (1992). *The Task-Artifact Framework: Representing Psychological Design Rationale with Claims and Scenarios*, IBM Watson Research Center, AAAI '92 Workshop on Design Rationale Capture and Use, San Jose, California, July 15, 47–55.

[Coad & Yourdon 91] Coad, P. & Yourdon, E. (1991). *Object-Oriented Design*. Yourdon Press Computing Series. Englewood Cliffs, NJ.

[Cockrel & Sander 92] Cockrel, L. & Sander, T.M. (1992). Selecting a Man/Machine Interface for a PLC-Based Process Control System. *IEEE Transactions on Industry Applications*, 28(4), 945–953.

[DeAntonellis & Bruna 90] DeAntonellis, V., & Bruna, Z. (1990). A Disciplined Approach to Office Analysis. *IEEE Transactions on Software Engineering*, 16(8), 822–827.

[Deignan 92] Deignan, P. (1992, June). Designing for a Complex World. *Apple Direct*, 8.

[Downton 87] Downton, A. C. (1987). Engineering the Man-Machine Interface. *Electronics and Power*, 691–694.

[Fujiwara & Kohno 85] Fujiwara, R., & Kohno, Y. (1985, June). User-Friendly Workstation for Power Systems Analysis. *IEEE Transactions on Power Apparatus and Systems*, PAS-104(6), 1370–1376.

[Halter 85] Halter, Richard. (1985). Man-Machine Interface Design Challenges. *Design News*, August 19, 63–70.

[Hammer 90] Hammer, Michael. (1990). *Reengineering Work: Don't Automate, Obliterate*, 18–26.

[Hopkin 89] Hopkin, V. D. (1989, November). Man-Machine Interface Problems in Designing Air Traffic Control Systems. *Proceedings of the IEEE,* 77(11), 1634–1642.

[Kettelhut 91] Kettelhut, M. C. (1991). Don't Let Users Develop Applications Without Systems Analysis. *Journal of Systems Management*, July, 23–26.

[Kloster & Zellweger 87] Kloster, G. V., & Zellweger, A. (1987). Engineering the Man-Machine Interface for Air Traffic Control. *IEEE Computer*, February, 47–62.

[Laurel 90] Laurel, B. (Ed.) (1990). *The Art of Human-Computer Interface Design*. Reading, MA: Addison-Wesley.

[Lehner 87] Lehner, Paul E. (1987). Cognitive Factors in User/Expert-System Interaction. *Human Factors*, February, 29(1), 97–109.

[Mayer, et al. 87] Mayer, R. J., et al. (1987). *Knowledge-based integrated information systems development methodologies plan* (Vol. 2) (DTIC-A195851).

[More 88] More, Roger A. (1988). Supplier/User Interfacing in the Development and Adoption of New Hardware/Software Systems:A Framework for Research. *IEEE Transactions on Engineering Management*, 35(3), August, 190–196.

[Visner 88] Visner, Robert J. (1988). Console Systems: The Man-Machine Interface. *Control Engineering*, September, 18–19.

[Zachman 87] Zachman, J. (1987). A Framework for Information Systems Architecture. *IBM Systems Journal*, 26(3), 276-292.