IDEF Family of Methods for Concurrent Engineering and Business Re-engineering Applications

Richard J. Mayer, Ph.D.
Knowledge Based Systems, Inc.
One KBSI Place
1408 University Drive East
College Station, TX 77840
(409) 260-5274

Capt. Michael K. Painter
Armstrong Laboratory/ HRGA
WPAFB, OH 45433
(513) 255-7775

Paula S. deWitte, Ph.D.
Knowledge Based Systems, Inc.
One KBSI Place
1408 University Drive East
College Station, TX 77840
(409) 260-5274

> "It must be remembered that there is nothing more difficult to plan, more doubtful of success, nor more dangerous to manage than the creation of a new system. For the initiator has the enmity of all who would profit by the preservation of the old institution and merely lukewarm defenders in those who would gain by the new ones" --
>
> Machiavelli, "The Prince," 1513 AD.

Introduction

In today's environment, system implementors in Corporate Information Management (CIM), Concurrent Engineering (CE), and Computer Integrated Manufacturing face two overwhelming challenges. Besides their primary responsibility for introducing a new system into their engineering and manufacturing organizations, they also have an underlying need to introduce a new system of processes for developing these implementations. These new system development processes must employ an integrated *framework*of modeling methods. That is, the development process uses a structured collection of methods, rules, procedures, and tools to support the development and evolution of systems. The framework guides the user in applying the appropriate method

within the system development life-cycle. The goal of this paper is to provide some insight into the purpose of modeling, particularly from the perspective of a CIM or CE project manager/engineer who must select, use, and evaluate the results of modeling efforts in support of systems development.

For all the rapid advances in computer hardware and specific software technology (e.g., databases), the bane of large-scale information systems development continues to be the lack of effective, well and widely understood methods for engineering such systems. With additional requirements for the system to be *integrated* and *evolving*, the complexities become truly overwhelming. A solid base of methods support is essential to these types of system development efforts. Complex, large-scale, evolving, integrated systems require multiple, diverse methods, each for a specific purpose. Thus, there is clearly a critical need for the development of effective methods. As a result, many methods remain to be developed.

A two-prong approach to method development is necessary. The goal of the first approach is to identify the methods needed to support an evolving, integrated information system (EIIS) and develop these methods. In the context of this paper, emphasis is placed on analysis and design methods needed to support EIISs for CE and CIM. Part of this method development includes developing precise, mathematical-based formalizations of the individual methods, as well as capabilities for translating among the methods.

The second approach is to develop an *engineering discipline* for the appropriate selection, use, extension, and creation of methods to support the planning, analysis, and design of large-scale, EIISs. That is, methods are developed that, in turn, are used recursively to develop new system engineering methods with predictable effectiveness. Within this approach, techniques for the analysis and comparison of methods must also be developed.

Developing methods (and particularly developing a methods engineering discipline) requires an uncommon experience base in method work. A method development team must include members with extensive experience in the actual application of the technology, as well as members with experience in methodization, language syntax design (both graphical and lexical), and formal models of semantics.

From a scientific viewpoint, the two interesting observations of methods appear to contradict each other. The first salient observation is that methods, almost by definition, are not derived in any logical, traceable manner; the second observation is that in spirte of this, *they work*. If the history of a method's development could be captured, it would be obvious that the method was not developed arbitrarily. Yet, methods do tend to be developed piece-meal, over time, by many individuals and as a result of a discovery process of what works well in a domain. Indeed, the underlying theoretical reasons for why a method works may not be understood at all by the discoverer. Rather, a hunch, an intuition, or an accidental circumstance may lead to such discoveries. For the very reason that a theoretical basis for a method may be unknown or not well understood, methods also tend to be difficult to enforce. It is simply difficult to convince someone to use a method when given no logical reasons, as the following anecdote illustrates.

> A young welder new to a railroad construction yard was assigned the job of forming and welding the stays for the large wooden barrels used as containers on the railroad cars. Proud of his welding prowess, the newcomer chafed that the foreman insisted on giving him instructions at great length on exactly how to carry out the job--instructions which the young welder considered outdated. He argued with the foreman that he could do the job much faster his own way. The foreman, for his part, insisted adamantly that the job must be done just this certain way or it wouldn't be done right.
>
> Unconvinced, the welder decided to show the foreman. In two days, he completed a week's worth of work which he triumphantly showed the foreman. Wordlessly, the foreman took one of the stays in hand, climbed to the top of the water tower, and threw the stay to the ground. The welder watched as the stay bounced twice and shattered at the seam. Climbing down from the water tower, the foreman quietly handed the welder a stay built the foreman's way and pointed to the tower. When the welder repeated the experiment with the foreman's stay, the stay hit the ground bouncing repeatedly, but holding firm.
>
> While the young welder may never understand why the foreman's method works, there is no doubt that he will use it religiously in the future. Whatever the exact nature of the foreman's particular method, it was typical of methods because it represented "best practice" in the domain of welding stays, a best practice learned over time and through experience. In fact, a method may be abstractly described as an encapsulation of best practice in a domain of cognitive or physical activity.

Nature and Importance of Methods

The purpose of a method is realized through its use by a human mind. Just as shovels themselves do not dig holes, but provide leverage for a human to dig holes, methods provide leverage for the human mind to accomplish a job more effectively. The method may assist and motivate the intellectual activities of the human, but it doesn't make the decisions, create the insights, or discover the problems.

Recognizing the nature of a method as an enabler, and not as a creator, should not diminish the recognition of the importance of methods. As the anecdote in the previous section illustrates, it may not be easy to pass down the knowledge of best practice from the expert to the novice. The basic importance of methods has long been recognized in the manufacturing industry. Methods are prominent in the "5 M's" of manufacturing--manpower, methods, materials, machines, and money. Materials, machines, and even money can be replaced, but manpower and methods that leverage the knowledge of the manpower are vital components of industry.

Components of a Method

Informally, a method is thought of as a procedure for doing something. That is, methods attempt to capture the "best practice" or experience. In addition, the method may have a representational notation to communicate this procedure more effectively. More formally, a method consists of three components as illustrated in Figure 1. Each method has a definition, a discipline, and many uses. The definition contains the concepts, motivation, and the theory behind the method. The discipline includes the syntax of the method, a computer interpretable format (labelled ISyCL [Mayer 91g] Syntax in Figure 1), and the procedure governing its use. Many methods have multiple syntaxes which have either evolved over time or are used for different aspects. Perhaps the most visible component of a method is the language associated with the discipline. Many system analysis and engineering methods use a graphical syntax to provide visualization of collected data in such a way that key information is unambiguously displayed. The use of a method may be by itself or within a suite of methods.

Types of Methods

The methods in the EIIS context are primarily methods that produce *models*, but some methods produce *descriptions*. Models and descriptions are similar in that they both consist of diagrams and texts. A model can be characterized as an idealized system of objects, properties, and relations designed in certain relevant respects within a particular structure to imitate the character of a given real-world system. The power of a model comes from its ability to simplify the real-world system it represents, and to predict certain facts about that system with corresponding facts within the model. Thus, a model is a designed system in its own right, constrained to satisfy certain conditions by the abstract system of which it is an instance.

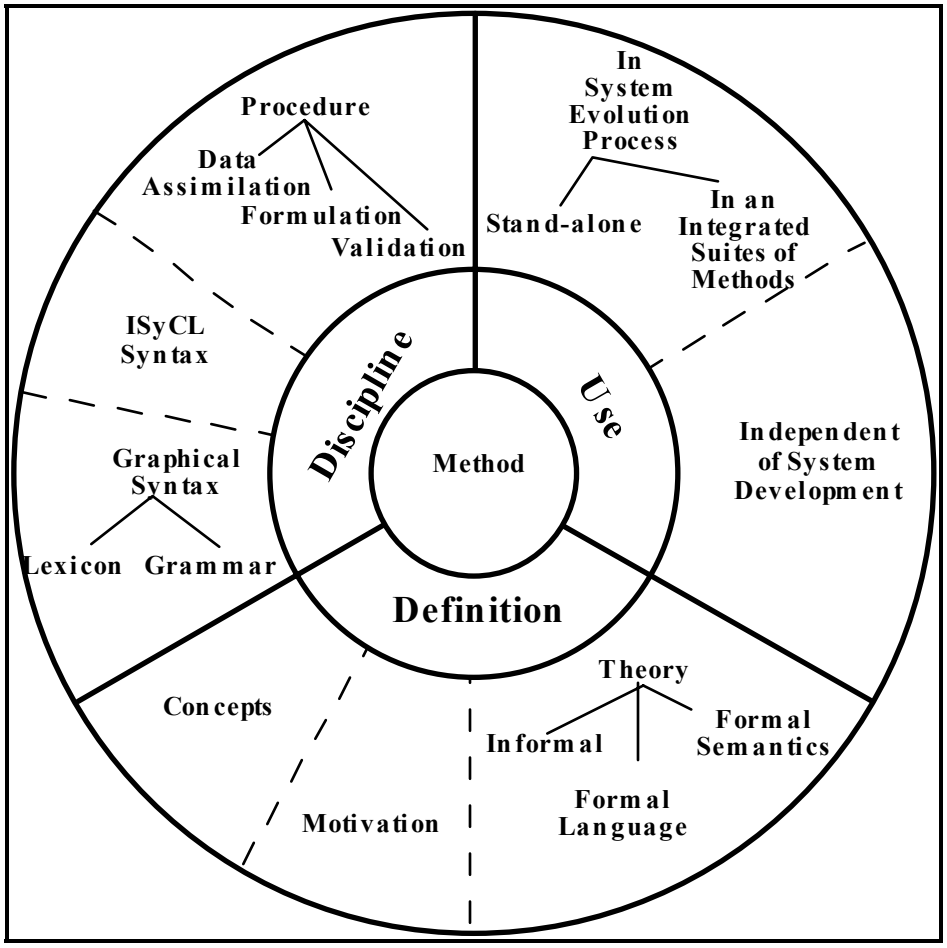

Figure 1. Components of a Method

Models are known to be incorrect, but assumed to be "close enough" to provide reliable predictors of the real properties of the domain of interest. A description, on the other hand, is a recording of facts or beliefs about the world. As such, descriptions are, in general, partial. A person giving a description may omit facts that don't seem relevant, or that he has forgotten in the course of describing the system. Thus, while descriptions must be accurate, they are not constrained by abstract, testable conditions that must be satisfied.

It is important to distinguish conceptually between models and descriptions. Unfortunately, the term *model* may be used ambiguously in a general sense to mean both

a description and a model. This occurs most frequently when discussing activities of modeling activities common to many methods and not a particular method. This paper discusses the family of IDEF methods including descriptive methods such as IDEF3 and IDEF5 and modeling methods such as IDEFØ, IDEF1. In this paper, the terms *model, modeler,* or *modeling* may refer either to models in the most general sense (referring to both models and descriptions) and models in the more restricted definition (i.e., versus descriptions). The context of the use of the term will clarify its intended meaning.

Methods in the System Development Process

All too often, modelers are caught between the system developer who claims that the effort cannot afford the lack of the modeling activity and the funding source who claims that the modeling effort is prohibitively expensive. In fact, there are many reasons to justify the time, labor, and expense required by a modeling activity to build system models or descriptions. A model of the system development process can be used to illustrate where system modeling fits. Figure 2 depicts the customer's perspective of the activities involved in developing a system and their relationships to one another. The nature of the system is unimportant and may be anything including night vision goggles, an information system, or even a house.

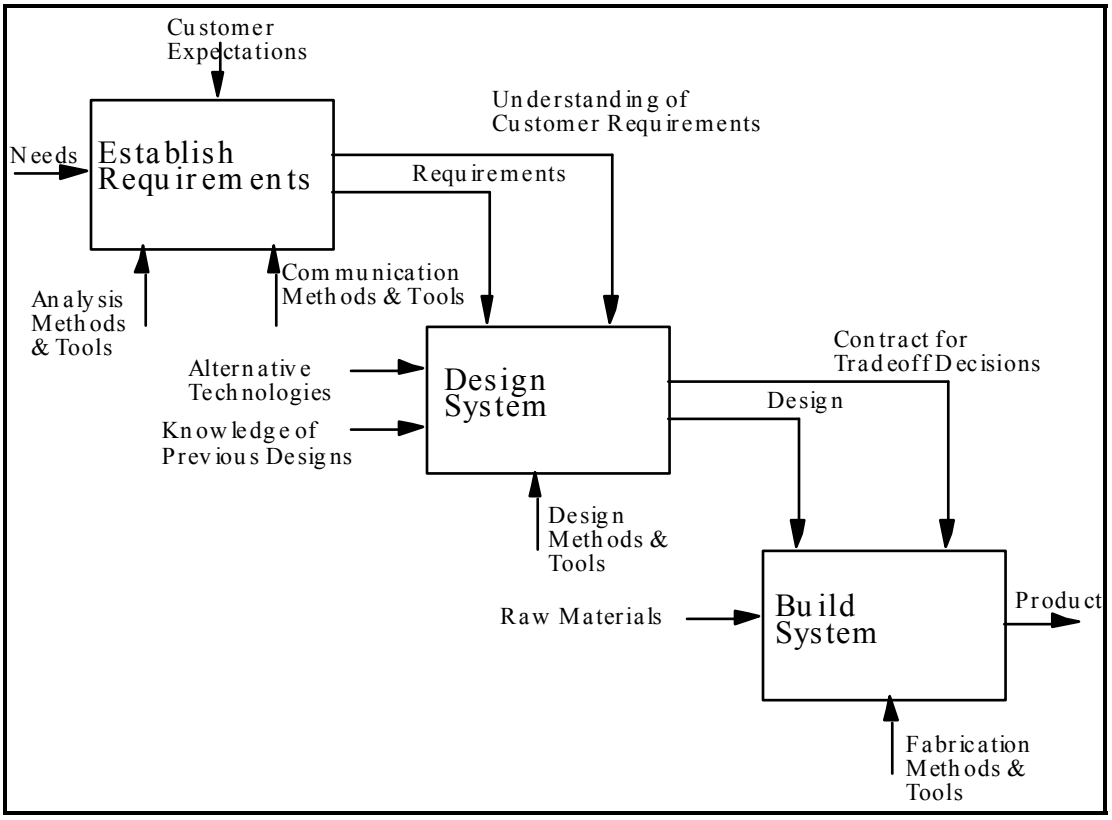

Figure 2. Develop System (IDEFØ Model)

In the activity labeled *Establish Requirements,* the customer provides input in the form of an expression of needs. While the customer may recognize problems in the environment, he/she usually does not understand the exact cause and may, in fact, misdiagnosis a symptom. *Needs* are conditions that must be satisfied for the problem to be considered solved. The system developer must establish the limiting constraints on the conditions to be satisfied, (i.e., the *requirements*). For example, an architect tasked with designing a new home will take an expression of needs for more space and lower heating bills and will ask questions that determine requirements.

How big is your lot size?

What kind of additional space do you need? Storage space? Sleeping space?

What is an acceptable range for heating bills?

How much money is to be spent on this house?

This activity produces two results: 1) a clear set of requirements and 2) a perception of the customer that the environment is understood well enough by the developer that the customers' needs will be satisfied. This perception is a by-product of the developers' attempt to isolate the causes of the problems through careful study of the environment where the symptoms occur. This analysis will not only lead to the rediscovery of the problems known to the customer, but will also often result in identifying other existing or potential problems.

As far as the customer is concerned, the developer can use any analysis method and tool as long as it accomplishes the goal and promotes lower costs and faster turn-around time. In this context, tools are considered support (usually automated) for using the prescribed methods. In fact, much generated by the method may never be directly seen by the customer. However, periodically, it is necessary to communicate to the customer what the system developer is actually thinking. This may amount to nothing more than asking the customer "Is this what you mean?"

The second activity, *Design System*, will typically not proceed without both a clear set of requirements and the customer's feeling that his expectations will be satisfied. The system developer then uses whatever design methods and tools will best help him satisfy the requirements. There may also be cases where communication methods and tools are used by the system developer to communicate with the customer throughout the design activity. An architect, for example, may generate a set of blueprints from the requirements without needing to consult further with the customer. Cases where it is necessary to involve the customer further will most likely occur when trade-off decisions have to be made. The system developer then presents the effects of competing design decisions in terms of constraints (e.g., cost). The trade-off decisions made at this point can then be captured either implicitly by verbal agreement or explicitly by formal sign-off.

Once the design is accomplished and the trade-offs have been accepted by the customer, the building of the system can begin. The fabrication methods and tools to be used in this process can range in sophistication from strictly manual approaches to totally automatic system generation.

Models built as part of the system development process, as a minimum, should 1) instill in the customer a feeling of assurance that the system developer understands the customer's environment, the customer needs, and the conditions that must be satisfied to meet expectations or the system requirements; and 2) involve the customer in making trade-off decisions and document those decisions. One important purpose for modeling from the customer's perspective is to satisfy these needs. A corollary to this assertion is that if the developer is building models that do not satisfy these expectations, success in meeting the customer's needs is left largely to chance. This should in no way overshadow the importance of models to system development needs. Rather, these guidelines should be used to help guide what kind and how much modeling is actually needed.

With the purpose of modeling from the customer's perspective more fully understood, the discussion will center on experiences in the use of the IDEF (Integrated Computer-Aided Manufacturing (ICAM) DEFinition) methods to perform modeling activities in support of CIM and CE system development. Experience with three such methods will be described in detail, namely, IDEFØ Function Modeling, IDEF1 Information Modeling, and IDEF1X Data Modeling. Following this discussion, the emerging IDEF methods including IDEF3 Process Description Capture, IDEF4 Object-oriented Design, IDEF5 Ontology Description, and IDEF6 Design Rationale Capture will be introduced and their envisioned application potential for CIM implementations described.

Function Modeling Using IDEFØ

The IDEFØ Function Modeling method is designed to model the decisions, actions, and activities of an organization or system. IDEFØ was derived from a well-established graphical language known as the Structured Analysis and Design Technique (SADT) [Mayer 90]. The Air Force commissioned the developers of SADT to develop a function modeling method for analyzing and communicating the functional perspective of a system. Effective IDEFØ models assist in organizing system analysis and promoting effective communication between the analyst and the customer. In addition, the IDEFØ modeling method establishes the scope of analysis either for a particular functional analysis or for future analyses from another system perspective. As a communication tool, IDEFØ enhances domain expert involvement and consensus decision-making through simplified graphical devices. As an analysis tool, IDEFØ assists the modeler in

identifying functions performed, what is needed to perform those functions, what the current system does correctly, and what the current system does incorrectly. Thus, IDEFØ models are often created as one of the first tasks of a system development effort.

Modeling Systems from an IDEFØ Perspective

IDEFØ includes both a process and a language for constructing a model of the decisions, actions, and activities in an organization. Applying the IDEFØ method results in an organized representation of the activities and the important relations between these activities in a nontemporal, non-departmentalized fashion. IDEFØ is designed to allow the user to "tell the story" of what an organization does. It does not support the specification of a recipe or process. Such detailed descriptions of the specific logic or timing associated with the activities requires the IDEF3 Process Description Capture Method. IDEFØ models isolate or separate *functions* from *organizations*, identifying common functional threads across organizational units, and facilitating organization-independent analysis.

IDEFØ has been successfully used as both an analysis tool and as a communication tool in a number of application areas. Referring back to Figure 2, this characterization indicates that IDEFØ can be applied as a mechanism for performing the *Establish Requirements* activity. The communication facilitation capability of IDEFØ makes it an effective analysis tool for cooperative interdisciplinary team projects as those required by any CIM or CE initiative.

Organizational Structures and Strategies of IDEFØ

A number of organization strategies designed in the IDEFØ method lend tremendous expressive power and ease in communication. When improperly used or not understood, they yield models that are difficult to comprehend or may make absurd declarations which appear well-founded. Examples of IDEFØ organization strategies include 1) the purpose, viewpoint, and context statements, 2) the hierarchical or top-down analysis approach to model development, and 3) the levels of abstraction.

For the modeler, these organization strategies focus work on one piece of the model and establish clear boundary conditions within which to perform the analysis. For the customer, they allow rapid discovery and inspection of the pieces of the system with

which the customer is most familiar. They also provide powerful browsing mechanisms for learning about the system as a whole and communicating the modeler's understanding of the system. The following will address the use of purpose, viewpoint, and context statements as organization strategies in the IDEFØ method. Following that discussion, the hierarchical or top-down analysis approach to model development and the notion of levels of abstraction will be discussed.

To begin an IDEFØ modeling activity, the modeler must first determine (and clearly describe) what the *purpose* of the model is, from what *viewpoint* the activity descriptions will be formulated, and within what *context*. The purpose is a statement of the goals of the modeling activities (e.g., what information needs to be assembled, what decisions this information is supposed to support, what consensus is to be achieved, etc.). For example, one purpose of an IDEFØ functional analysis could be to identify opportunities for consolidating existing functions under a new CIM strategy. An accepted purpose provides the modeling team with a completion criteria. That is, when the purpose is accomplished, the model is finished.

The viewpoint statement describes the perspective that should be taken when constructing, reviewing, or reading a model. This viewpoint establishes how the reader will interpret the model and how the modeler will constrain his idealization or abstraction of the activities that occur in the system under study. An accepted viewpoint statement provides the modeling team a mechanism for controlling the scope and level of detail in a model.

The context establishes the interpretation and scope of the model as part of a larger scope. This focus creates a boundary within the environment for the model.

Another strategy for organizing the development of IDEFØ models is the notion of *hierarchical decomposition* of activities. Although IDEFØ models are developed using a hierarchical or top-down approach, Doug Ross, the creator of SADT, is thought to have often struggled with these terms. In fact, Mr. Ross proposed that this process might more accurately be characterized as an Outside-in approach [Ross 85]. A *box* in an IDEFØ model, after all, represents the boundaries drawn around some activity. Looking inside that box leads one to discover the breakdown of that activity into smaller activities which together comprise the box at the higher level.

This hierarchical structure helps the analyst keep the scope of the model within the boundaries represented by the activity's decomposition. The customer also finds this organization strategy useful for hiding unnecessary complexity from view until a more in-depth understanding is required by looking inside the box at its decomposition (See Figure 3).

This complexity hiding may also be characterized as part of the abstraction mechanism used in IDEFØ. One common misconception, however, is that levels of abstraction are only evidenced in the activities themselves as one moves between levels of the model. The arrows also exhibit different levels of abstraction between levels of the model. In fact, achieving the correct balance between the level of abstraction associated with a box and the level of abstraction associated with the arrows attached to the box is not always trivial. For example, suppose the four mechanism arrows inside the inner box in Figure 3 represent different types of tools used to accomplish the their respective activities. These four arrows could be bundled together into a more abstract perspective as a single arrow labeled "tools." Thus, the outer box in Figure 3 would have only one mechanism arrow for its level of abstraction. A far less elegant depiction would have all four arrows appear at both the more abstract and the more detailed levels of the model. It can be seen that one easy way to tell whether or not the modeler has effectively used the information hiding constructs available in IDEFØ is to count the number of arrows attached to the boxes at any given level. If the model seems cluttered with arrows, it is very likely that the level of abstraction used in bundling the arrows is not the same as the level of abstraction as the activity.

Perhaps the least understood and most frequently misapplied IDEFØ constructs for screening unnecessary detail at a given level of abstraction is the notion of bundling and unbundling of arrows. It would be logically inconsistent to unbundle all but two of the four mechanism arrows in Figure 3 when each occurs at the same level of abstraction, namely, as component tools. Likewise, it would make little sense to unbundle and rebundle an arrow at the same level of abstraction. Unfortunately, the IDEFØ literature does not adequately cover how to appropriately avoid logical inconsistencies that can be introduced through incorrect use of information hiding constructs through arrow bundling. The best approach is to build models using an automated support tool that

enforces good practice. Otherwise, an IDEFØ modeler can take years to learn how to recognize and avoid bundling problems.

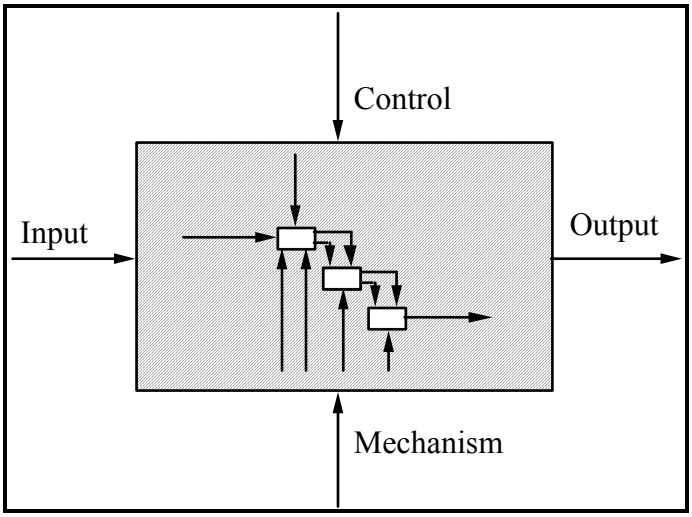

Figure 3. Looking Outside-In

IDEFØ Guidelines

There is a considerable knowledge base of heuristics available for aspiring IDEFØ modelers [Cullinane et. al. 90, KBSI 90a, Mayer 89b, Softech 81a]. In the following paragraphs, a few of the more common hints for obtaining the most from an IDEFØ modeling exercise are discussed. Specifically, these guidelines will help determine how well a given model satisfies the purposes of modeling outlined above.

Without question, the most difficult aspect to master in IDEFØ modeling is maintaining the same consistent purpose and viewpoint between levels of the model. What makes this task so difficult is the difficulty of recognizing shifting viewpoints. A good heuristic is to look at the boundaries within which the modeler develops a decomposition and formulate questions including the following: "Does this activity fall within the scope of the higher-level activity?" and "Does this activity conform to the established viewpoint and purpose of the model?"

Secondly, look for models that push the numerical constraints of the method. For example, the discipline component of the IDEFØ method establishes a rule that there should never be less than three nor more than six activities to a decomposition. Likewise, there should never be more than six arrows on one side of an activity box. A common modeling mistake is to decide that a seventh activity in the decomposition becomes indispensable. In fact, one tendency is first to draw six boxes for the decomposition and then attempt to come up with names for all six activities. Another mistake occurs when activity boxes and their associated arrows begin to look like wiring diagrams for electronic components. This is largely due to a failure to logically organize the arrows into bundles at different levels of abstraction consistent with the same level of abstraction as their associated activity boxes. Inappropriate bundling, such as that done simply to abide by the established rules, may also occur. These kind of errors become obvious when arrows seem arbitrarily grouped together.

Another common problem emerges when new conventions are introduced into the method. For example, some modelers will choose to establish a convention that inputs and outputs can only be data elements. In this way, they hope to ensure that inputs and outputs translate directly into information model elements. Intuitively, this approach would then provide clear and unambiguous tracking of data needs across activities as well as clearly delineating the scope of their information models. However, with this approach, the modeler is forced to further change conventions by making mechanisms into resources assigned to an activity. For example, consider an activity *Fix Broken Airplane*. With these conventions, there is only one place for the broken airplane, as an input to the activity. But with this approach, there is no place to show that what emerges is a fixed airplane, since outputs can only be data elements. It is far more useful to use the existing conventions and then use inputs and outputs as candidates for what may be data elements to be examined and discriminated later.

A frequently misunderstood convention of IDEFØ is the built-in notion of nontemporality implicit in the IDEFØ method. As discussed previously, IDEFØ does not explicitly capture time-ordered constraints between activities. In fact, temporal logic is purposely not included in IDEFØ to lend more expressive power and generality. While IDEFØ could provide a description of a specific set of activities operating within the bounds of a specific time-ordered process, it is far more useful for analysis purposes to

provide a generalized model which accounts for all, or at least all relevant, paths that could be taken through a set of activities for any number of time-ordered processes. Models should therefore be judged, in part, by the degree to which they accommodate likely or possible sequences of activities and should not be built with the intention of implying a specific process.

Perhaps the most useful exercise used for assessing the quality of an IDEFØ model is to sit down, read the model, and determine if it makes sense with a constraint of no more than two minutes per page. For a large model, this should require no more than two hours. If a reader can understand the environment modeled by the IDEFØ representation within that time and feel capable of explaining what occurs in that environment, it is likely that the model is of significant value. Models that require two full days of careful study to fully comprehend are not good IDEFØ models.

Information Modeling Using IDEF1

Referring to Figure 2, IDEF1 is viewed as a method for both analysis and communication in establishing requirements. In this case, however, IDEF1 establishes the requirements for what information is or should be managed by enterprise. In CIM applications, IDEF1 is generally used to 1) identify what information is currently managed in the organization, 2) identify which of the problems identified during the needs analysis are caused by lack of managing appropriate information, and 3) specify what information will be managed in the "TO-BE" CIM implementation.

The IDEF1 information modeling method derives its foundations from three primary sources. the Entity-Link-Key-Attribute (ELKA) method developed by Hughes Aircraft, the Entity-Relationship (ER) method proposed by Peter Chen, and Codd's Relational Model (see Figure 4). The original intent of IDEF1 was to capture what information exists or should be managed about objects within the scope of an enterprise. The IDEF1 perspective of an information system includes not only the automated system components, but also non-automated objects such as people, filing cabinets, telephones, etc. IDEF1 was specifically designed to not be a database design method. At the time of IDEF1 development, the database community believed that a method for analyzing and stating information resource management needs and requirements was needed. This was

the intent of IDEF1. Rather than a design method, IDEF1 is an analysis method used to identify the following.

1. The information collected, stored, and managed by the enterprise.
2. The rules governing the management of information.
3. Logical relationships within the enterprise reflected in the information.
4. Problems resulting from the lack of good information management.

The results of information analysis can be used by strategic and tactical planners within the enterprise to leverage the information assets to achieve competitive advantage. Part of these plans may include the design and implementation of automated systems which can more effectively take advantage of the information available to the enterprise. IDEF1 models provide the basis for those design decisions. IDEF1, then, is not used to design a database; rather, it is used to provide managers with the insight and knowledge required to establish good information management policy. The next section will provide an overview of some of the basic concepts and rules of IDEF1. The interested reader is referred to [Softech 81b, KBSI 90b, Mayer 89b, Menzel 89] for additional details regarding this method.

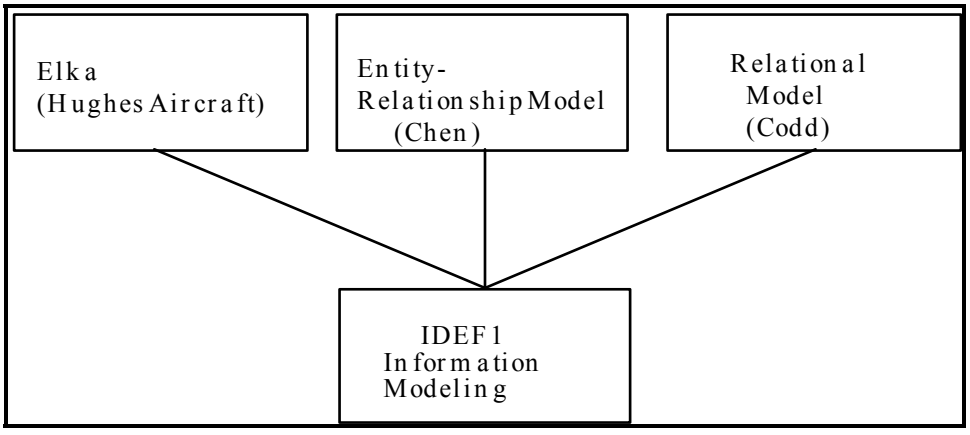

Figure 4. IDEF1 Origins

Modeling Systems From The IDEF1 Perspective

IDEF1 uses simple graphical conventions to embody a powerful set of rules which help the modeler distinguish between the following.

1) Real-world objects.

2) Physical or abstract associations maintained between real-world objects.

3) The information managed about the real-world object.

4) The data structure used to represent that information for acquiring, applying, and managing that information.

Simply stated, IDEF1 was designed to represent information that is, or should be, collected, managed, controlled, and ultimately paid for by the enterprise (item 3). The rules of the method help prevent modeling items 1 and 2 (normally considered the province of knowledge engineers). They also divert the attention of the modeler away from database design (item 4, normally considered the province of software engineers).

There are two important realms to modelers in determining information requirements. The first realm is the real-world as perceived by the people in an organization. This realm includes the physical and conceptual objects (e.g., people, places, things, ideas, etc.), the properties of those objects, and the relations associated with those objects. The second realm is the information realm. This realm includes information images of those objects found in the real-world. An information image is not the real-world object, but only the information collected, stored, and managed about real-world objects. IDEF1 is designed to assist in discovering, organizing, and documenting this information image. These tasks are essential to any CIM implementation. This does not mean that the task of structuring the organization's knowledge of the first realm is not important. Looking towards intelligent CIM systems and/or CIM implementations that embody large numbers of knowledge based systems, this task (often referred to as *ontology* definition) becomes more important. However, a specialized method, IDEF5, is designed to assist in the structuring of this knowledge base. IDEF5 and its application are discussed in a later section. The following section concentrates on the role of IDEF1 which attempts to capture an organization's information management requirements.

IDEF1 Basic Concepts

Focus on the real-world realm for a moment (see Figure 5). The term *real-world object* is used to describe real-world people, places, things, or ideas. In this sense, the sales department in a company may be a real-world object, as is an employee working in that department. These objects have characteristic properties associated with them, such as a name, age, gender, etc. Further, one real-world object may be involved in some kind of association with other real-world objects. For example, an employee may *work for* a department.

Now, focus on the information realm. An IDEF1 *entity* represents the information maintained in a specific organization about physical or conceptual objects (e.g., people, places, things, or ideas). For example, an IDEF1 entity exists when an organization maintains information about the sales department resulting in the existence of an *information image* of that object in the organization's information system. In IDEF1, the term *entity class* refers to a collection of entities or the class of information kept about objects in the real-world. An *entity class* can be thought of as an empty box for holding 3" x 5" cards with each card an actual entity. The box is labeled on the outside with 1) an entity class name describing what type of cards go in the box, and 2) a template for the individual cards that will eventually go inside.

Entities have characteristic *attributes* associated with them that record values of properties of the real-world objects. Using the card file model, an *attribute class* is the template for the attribute-value pairs found on the individual file cards. The term *attribute class* refers to the set of attribute-value pairs formed by grouping the name of the attribute found on the outside of the file box, and the values of that attribute for individual entity class members (entities), listed on the individual cards themselves. A collection of one (or more) attribute classes which allow us to distinguish one card from another, or one member of an entity class from another, is called a *key class*. A key class is indicated by placing it in the top left corner of the template and underlining it.

A *relation* in IDEF1 is an association between two individual information images. The existence of such a reference is discovered (or verified) by noting that the attribute classes of the one entity class contain the attribute classes of the key class of the referenced entity class member. For example, the information managed about an employee may contain a department number (an attribute class that belongs to the collection of information kept about a department). A *relation class* can be thought of as

the template for associations that exist between entity classes. An example of a relation class in IDEF1 would be the label *works for* on the link between the information entity class *employee* and the information entity class*department*. It is important to note that if no information is kept about an association between two or more objects in the real-world, then from an IDEF1 point of view, no relation exists. Relation classes are represented by links between the entity class boxes on an IDEF1 diagram. The diamonds on the end of the links and the half diamonds in the middle of the links encode additional information about the relation class (i.e., cardinality and dependency). These links often indicate the existence of a business rule of an organization. If there are inconsistencies during the analysis of these links, the analyst very often discovers inconsistencies in business rules.

The procedure portion of the IDEF1 method was designed to be scalable from small department level analyses to large enterprise-wide projects. It has been demonstrated as an effective problem solving method where the cause of a particular problem has to do with the lack of management (or mismanagement) of information. However, it is important to have the analysis done correctly. The next section describes some indicators of quality in an information analysis performed using IDEF1.

IDEF1 Modeling Guidelines

There are some very simple ways to quickly inspect an IDEF1 model and determine whether or not the modeler has been able to take the expression of need, together with an understanding of the environment, and successfully identify existing and future information management requirements. These inspection techniques will help the customer identify, almost immediately, whether or not the IDEF1 modeler is modeling from the real-world realm (generally considered incorrect) or from the information realm (generally considered correct).

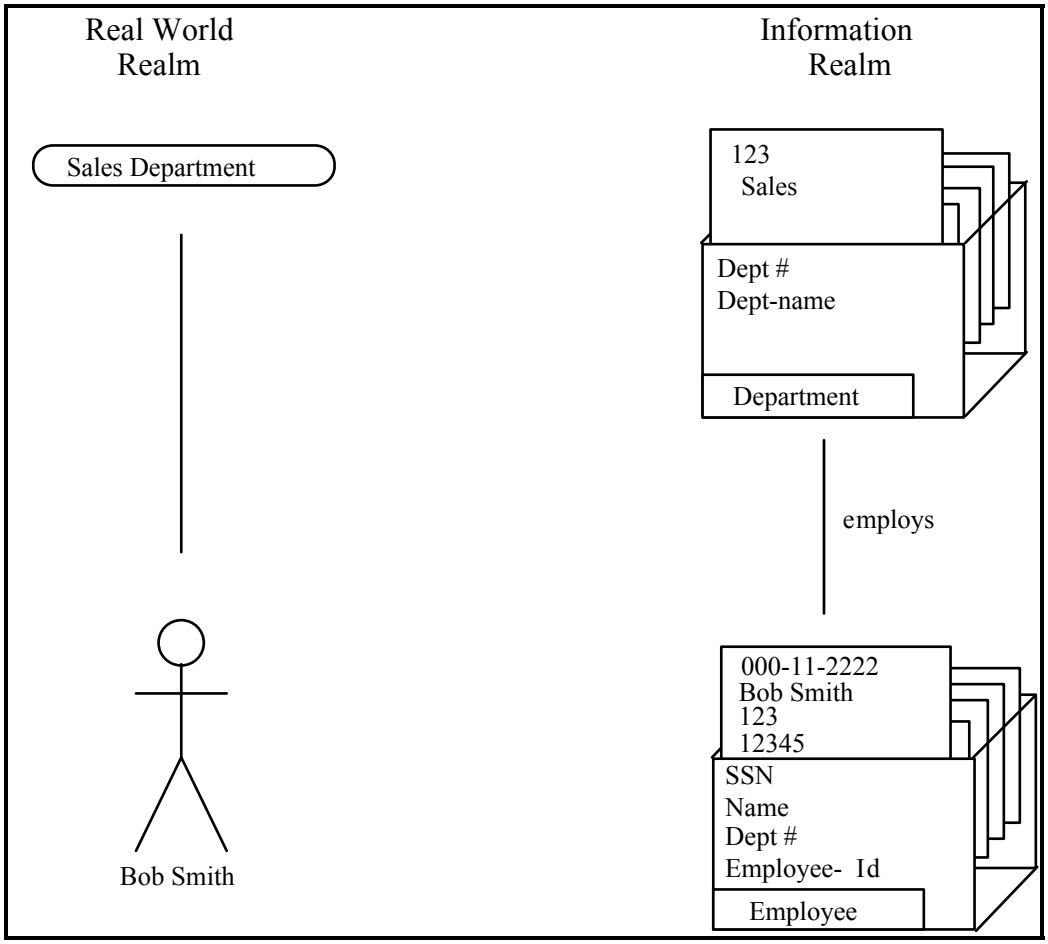

Figure 5. IDEF1 Basic Concepts

First, check the labels used to name the entity classes. If the labels are plural rather than singular names, it is unlikely that the model was created with the information realm in mind. For example, a box labeled *Employee* conforms better to the convention that the entity class represents the set of information important to manage about a real-world person than a box labeled *Employees*. With the *Employees* label, the modeler may mean a group of people with some of the same properties (i.e., individuals who all have a job at the company). Remember, an IDEF1 modeler is not concerned about the real-world objects directly, only the information about them that is actually managed by the organization.

A second indication of models which represent real-world objects and not information images are models which have some boxes without non-key attribute classes and others with whole laundry lists of them. Those models without non-key attribute classes typically are objects like organizational units. Close inspection of boxes with long lists of attributes will often reveal that the box represents a form, like a *Purchase Order* form, with the attributes the actual fields on the form.

The third check is to look for models that have most of their entity boxes with only one or two relations whereas a few entity boxes are wired with many relations. In this case, it is highly probable that the box with all the relations will be the entity box with a label such as *Person*. In this example, many of the relations attached to the enetity box indicate the roles that a person can assume. For example, the model might read: Person inspects Parts; Person certifies Inspection; or Person is married to Person.

Although it is a natural tendency for first-time IDEF1 users to model what they know about the real-world in terms of what can be easily seen, misapplication of IDEF1 may lead to huge models with little or no benefit. With little direction, the novice modeler attempts to model everything that is observed in the real-world. These models become more and more difficult to manage as they continue to grow. Soon, it becomes easy to lose sight of the purpose of the modeling exercise. Information models that model the real-world realm rather than the information realm provide virtually no insight into what the information requirements are or where more effective information management policies can improve competitive posture.

Guidelines When Using IDEF1

A potential trap for information system developers is to assume that the ability to uniquely identify one information entity from another necessarily implies that one real-world object can be uniquely identifiable from another. Remember, an information model represents information that is actually managed about real-world objects. Any other knowledge is unavailable to the information system. When information is kept about individual real-world objects, the information model can be used as a means for identifying specific real-world objects. For example, the serial number used to distinguish one engine from another may also serve to distinguish one entity called *Engine* from another. However, when information is kept only about types of real-world

objects, the same situation does not hold. Consider, for example, a standard bolt. A unique part number is not usually assigned to each bolt. The information model, however, may use part number to uniquely identify the information kept about specific families of bolts.

To avoid confusion , the IDEF1 modeler should be careful naming the relation classes and the links that connect entity class boxes. As seen in Figure 5, it seems very intuitive to read the model as, "A Department employs one or more Employees". Is this the same as, "The information kept about Departments employs the information kept about Employees"? Obviously, the first reading may confuse one into thinking that the file box is actually the real-world object and not the information kept about an object. Remember, the link between the boxes represents a reference from one file card to another, or information relationships, not real-world relationships.

A sentence such as "A Department employs one or more Employees" is a natural language fact or a business rule. Since business rules are important to know and manage, perhaps they should be tracked and managed in a similar fashion to how source data items in IDEF1 are tracked and managed. The only exception being that they are managed separately as a source facts list. In the meantime, the links between entity classes can be labeled L1, L2, etc. This makes it very clear that a relation class represents a function which when applied to a member of one entity class will return those entity class members involved in the information relationship. Information relationships and real-world associations cannot be indiscriminately mixed without leading to confusion, paradox, and error [ISO 87]. For example, suppose an entity class *System* is created and attached to a relation originating from and ending at the *System* entity class with a *is comprised of* relation class label. How is such a model interpreted?

Intuitively, it seems reasonable to interpret the entity box as representing a system/subsystem hierarchy. The highest level system may be something like an automobile that is comprised of subsystems and further subsystems to the individual component level. With this approach, it appears that the same system/subsystem hierarchy can be ascribed to the information kept about a vehicle to enable more intuitive thinking. This approach implies that the information kept about the highest level system is comprised of the information kept about subsystems below it in the system hierarchy. However, this doesn't work because the information kept about an automobile includes

information such as who owns it, what state it is registered in, what year it was manufactured, etc. None of that information has anything to do with the information kept about subsystems such as the braking system, which might include the type of front and rear brakes, the moisture content in the brake fluid, or the thickness of the remaining brake pad. Obviously, information relationships will not behave in the same manner that real-world relationships behave. The only correct interpretation from an IDEF1 viewpoint is that one member of the class of information kept about an object can reference information about other objects through the referenced member's key class.

The rules and procedures of IDEF1 assist the construction of accurate models of information currently managed in an organization targeted for CIM implementation. They also serve as mechanisms for definition of the information requirements for the TO-BE system. However, IDEF1 was designed to be technology independent. Therefore, when the design for the CIM implementation is being started, two other IDEF methods are used: IDEF1X for relational data base implementations and IDEF4 for object-oriented implementations. These methods are described in the following sections.

Data Modeling Using IDEF1X

In Figure 2, IDEF1X is intended as a method for accomplishing the *Design System* activity. Because it is a design method, IDEF1X is not particularly suited to serve as an "AS-IS" analysis method, although it is often suggested as an alternative to IDEF1. IDEF1X is most useful for logical data base design after 1) the information requirements are known and 2) the decision to implement using a relational database has been made. Hence, the IDEF1X perspective of the information system is focused on the actual data elements in a relational database. If the target system is not a relational system, e.g., an object-oriented system, IDEF1X is not the best method. The interested reader is referred to [GE 85, KBSI 90c, Mayer 89b] for additional details regarding this method.

The development of IDEF1X was influenced by Chen's Entity Relationship (ER) model, Codd's Relational model, and Smith's Aggregation/Generalization model. These origins led to the development of the Logical Database Design Technique (LDDT) by the Database Design Group, Inc. LDDT later became a commercial product of Dan Appleton Company (DACOM) as the Data Modeling Technique (DMT). A coalition of companies lead by General Electric including SDRC, CDC, and DACOM used this

method and their experience with IDEF1, to create the IDEF Data Modeling Method, or IDEF1X (See Figure 6).

Modeling Systems From The IDEF1X Perspective

Once there is a thorough understanding of the information requirements, decisions about how to manage that information more effectively can be made. One possible decision is to implement an automated system, requiring the selection of an appropriate design method. Good design methods should result in a robust, high quality, cost effective implementation with affordable life-cycle costs. Therefore, a good design method must embody the best application experience associated with a particular technology. This usually means that a design method will generally work "well" only for that technology whose experience base it encapsulates. The IDEF family of methods recognizes this fact and provides specific methods for design with particular implementation technologies in mind. If the technology of choice is the relational database technology, IDEF1X is appropriate for logical database design. If the technology of choice is an object-oriented database paradigm, IDEF4 (discussed in a later section) is more appropriate.

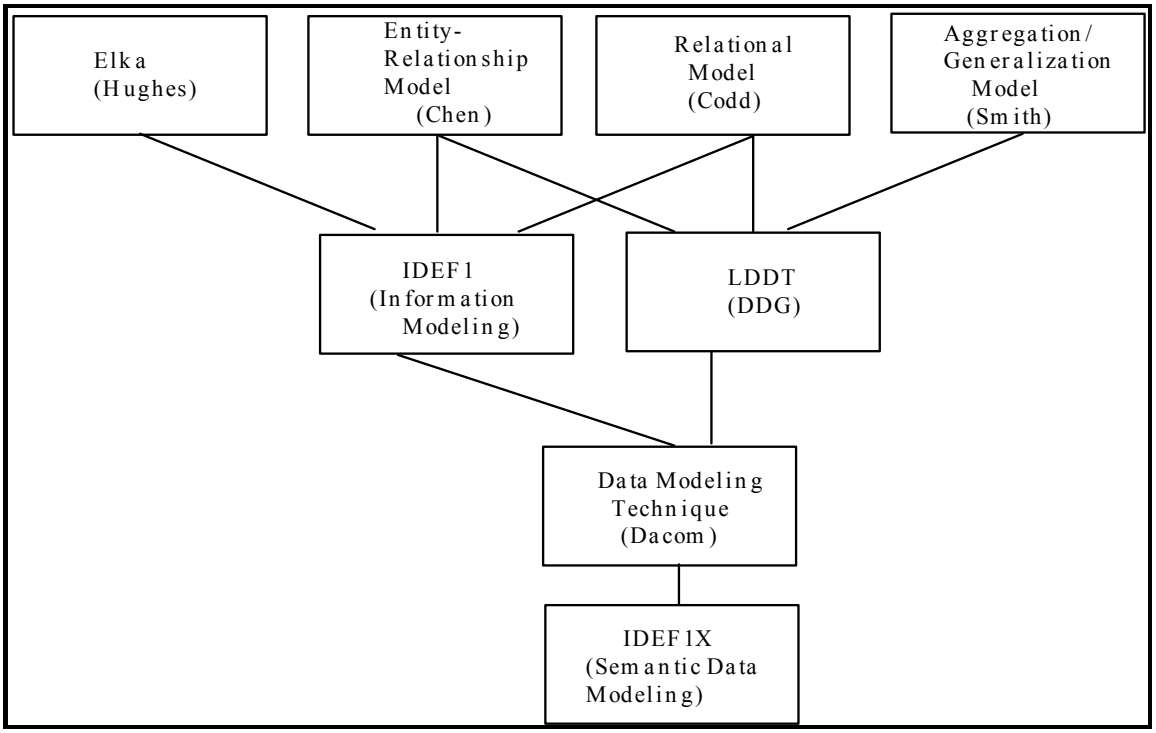

Figure 6. IDEF1X Origins

There are several reasons why IDEF1X is not well suited for non-relational system implementations. IDEF1X requires that the modeler designate a key class to distinguish one entity from another, whereas object-oriented systems do not require keys to individuate one object from another. In situations where more than one attribute or set of attributes will serve equally well for individuating IDEF1X entities, the modeler must designate one as the primary key and all others as alternate keys. The explicit labeling of a foreign key, which is an attribute owned by one entity, but which serves as the key attribute in another entity, is also required. Modeling constructs like these clearly indicate specific intent to incorporate the best practice in logical design for relational systems implementations. The results (logical design models) of an IDEF1X activity are intended to be used by the programmers involved in taking the "blueprint" for an information system, or the logical database design, and implement that design in a relational database. However, the modeling language of IDEF1X is sufficiently similar to that of IDEF1 that the designs generated from the information requirements specification can be easily reviewed and understood by the ultimate users of the proposed system.

IDEF1X Basic Concepts

Since the terminology used in the IDEF1X method is very similar to that used by the IDEF1 method, further definition of terms is necessary to avoid confusion. There are fundamental differences in the theoretical foundations and concepts employed by the two methods. An *entity* in IDEF1X refers to a collection, or set, of similar data instances (data records about persons, places, things, or events) that can be individually distinguished from one another. Thus, an entity box in IDEF1X represents a set of data items in the real-world realm. An *attribute* is a slot value associated with each individual member of the set, (these individual members are called *entity instances*). In Figure 7, the record labeled *Bob Smith* and *Sales Department* are both entity instances. *Department* is the collection of specific records in a relational table representing departments; *Employee* is the collection of records about people employed by individual departments. *Department-Name*, *Department Number*, and *Department-Location* might be attributes of the *Department-Entity*. The *relationship* that exists between individual members of these sets is given a name. In this case, this relation is interpreted as establishing a *referential integrity constraint*.

A powerful feature of the IDEF1X method is the support for modeling logical data types through the use of classification structures. This classification structure is the *generalization/specialization construct*. This construct is an attempt to overlay models of the natural *kinds* of things that the data represents whereas the boxes, or entities, attempt to model *types* of data things. These *categorization relationships* represent mutually exclusive subsets of a generic entity or set. In other words, subsets that emerge from the superset cannot have common instances. One example of how this is used is to state that given a generic entity *Person*, two subsets can be created which represent a complete set of categories of people, namely, *Male* and *Female*. No instance of the *Male* set can be a instance of the *Female*set, and vice versa. The unique identifier attribute for a instance of the male set is, by

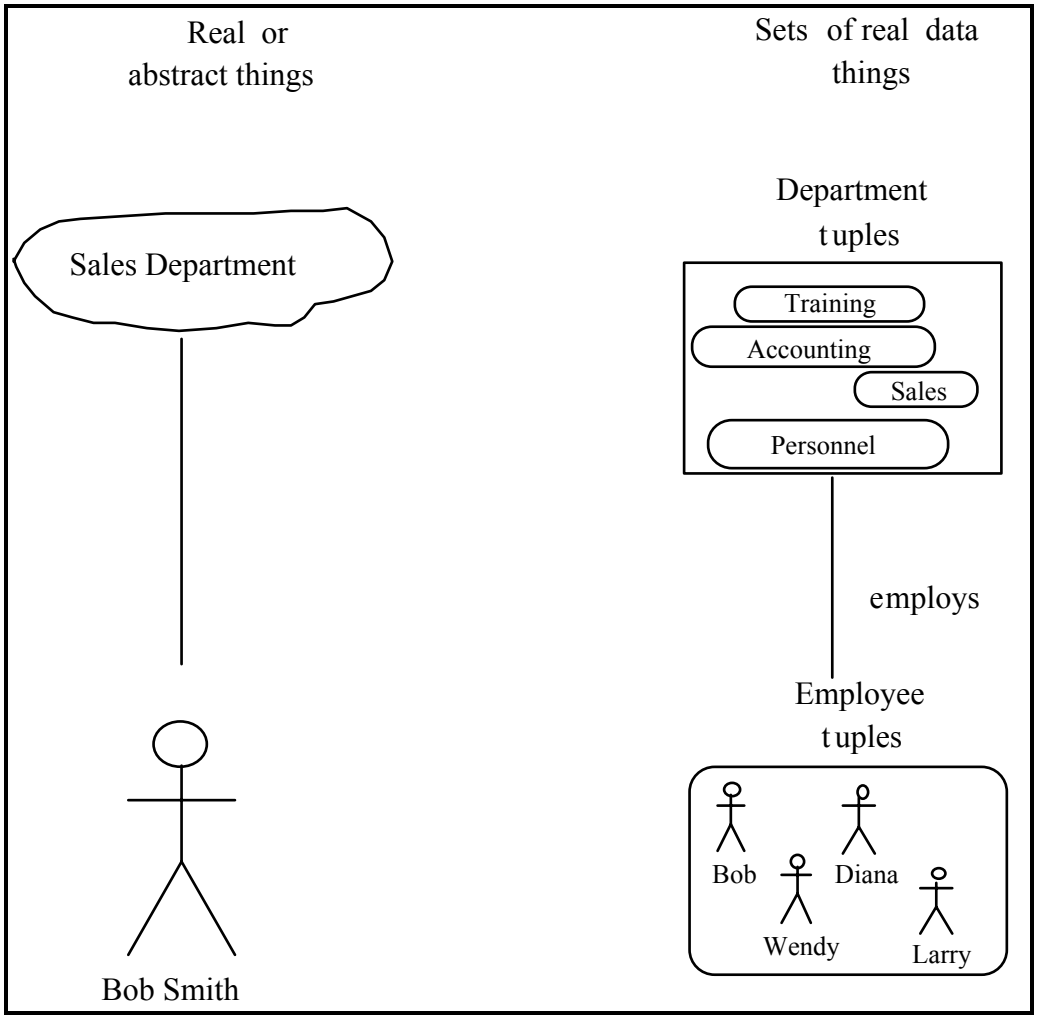

Figure 7. IDEF1X Concepts

definition, the same type of data as that for a instance of the generic entity. The same holds true for the female category entity. The general attributes that apply to all members of the entity *Person* are listed in the generic entity. The specialization attribute, gender in this case, is listed in the category entity.

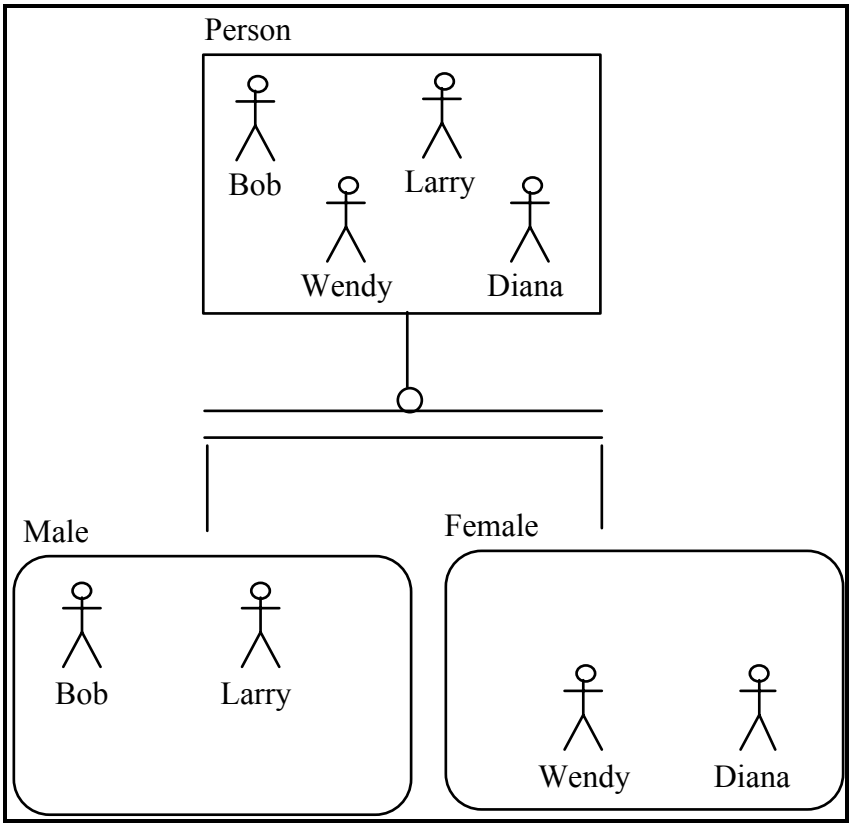

Figure 8. IDEF1X Generalization/Specialization Construct

Guidelines When Using IDEF1X

The underlying concepts associated with the IDEF1X method are intended to bridge the modeling of natural language facts about real-world things, people, places, events, etc., with the modeling of logical data structure This is very different than the IDEF1's goal which focuses strictly on the information image of the real-world things (not the things themselves, nor the data structures that represent the information about the things). IDEF1X is often extended to go a step further by also attempting to design data specifications associated with these sets of data items. This is accomplished through the glossary associated with the attributes and relationships.

Considering the examples discussed in the section on IDEF1, the intent of IDEF1X is to model how the database represents facts such as "A department employs one or more

employees" or "A system is comprised of one or more subsystems." Since real-world relationships between objects do not behave the same way that information relationships do, it is questionable if this approach is useful for designing logical data structures. The reader may want to address this question by reconsidering the example of the model representing hierarchies of systems. From considering this example, one can easily recognize ample possibility for misuse of the IDEF1X capabilities exists. This misuse results in models of the real-world things, and not sets of relational database entities.

Another example of this problem is illustrated in an entity called "Technical Order Improvement" that appeared in a model delivered to the Air Force intended to establish the specification for a logical database design. This entity name would have been more accurate with the word *Form* appended to the end of the entity name. This is because the attributes listed in the entity actually correspond to individual fields on an Air Force form called the AFTO 22. The entity here is used to model an artifact (namely the form itself) which is a list of attributes.

The first question to ask is, "Is this a reasonable model of the set of forms called AFTO 22s?". This entity certainly presents an accurate model of the fields on the existing form and may therefore be fully justified. The next question to ask is, "Has any design activity been facilitated through modeling the form in this manner?". In other words, "Does this kind of modeling assist in moving from a set of information requirements to the logical design of relational tables and from which trade-off decisions can be made?". Does it make sense to create a table, or relation, in a relational database that corresponds directly with what one would find on existing forms? This approach may have little or no impact for small implementations where only a limited set of artifacts have to be modeled. However, for large-scale implementations, the plethora of different forms would easily employ an army of modelers maintaing the inventory of the forms and their associated fields. More importantly, if such a model is handed to an implementor as the design specification, the implementation will be a simple computerization of the current paper system. Such automation of paper systems are infamous in the history of CIM failures.

Based on these examples, it is obvious that misapplication of a design method can lead to confusion and/or poor design. One possible solution is to establish the convention that an entity must represent information kept about real-world objects rather than the objects themselves. But, this would result in little more than a syntactic variant of IDEF1, with

one major exception, namely, the generalization/specialization construct. This too will lead to confusion if used without exercising great caution. For example, is it true that the set of information kept about people is divided into two types, namely, male information and female information? Probably not, and if so, it would probably be illegal. More importantly reducing IDEF1X to IDEF1 loses a good design method.

Perhaps, IDEF1X is best used as a method for modeling just the objects themselves and not the structure of the data associated with those objects. Models of this type are often called *concept models*. Again, the difficulty occurs with the generalization/specialization construct since in the real-world, kinds are usually overlapping, and not mutually disjoint. Since IDEF1X does not allow for categorization entities with mutual members, IDEF1X could not serve well as a concept modeling language. For example, a model of the kinds of employees found in a company might include managers, engineers, designers, and secretaries, among others. A generalization entity in this case would be *Employee*, and the specialization entities would be those previously listed. The interpretation of this structure implies that engineers cannot be designers, nor can they be managers.

Methods are needed for three different aspects. First, methods are needed that can effectively capture what is known about the real-world and the relationships that exist between people, places, events, etc. Second, there is a need for methods that can capture existing and anticipated information management requirements. Third, good methods for supporting the design of systems that apply specific technology to meet the information management requirements are needed. The IDEF3 Process Description Capture and IDEF5 Ontology Description methods discussed in the following sections are specifically designed to address the first of these needs. The IDEFØ Function Modeling and IDEF1 Information Modeling were specifically targeted at the second of these needs. The IDEF1X Data Modeling, IDEF2 System Dynamics Modeling, and IDEF4 Object-oriented Design were developed to satisfy the third need. Unfortunately, there are a number of commonly used modeling languages which fail to maintain an unambiguous distinction between these three realms.

System Dynamics Modeling With IDEF2

The potential for benefits from the use of simulation modeling for manufacturing and information system analysis, planning, and design have been well established over the past 20 years. As a result, more and more decision makers turn to simulation to study the complex interactions and the dynamic behaviors of integrated manufacturing systems. Simulation modeling is a powerful decision support tool that aids in solving complex problems in a variety of application domains. Simulation is useful when:

1. The cause of a problem is the result of a complex time dependent interaction among the components of the system.
2. The effect of a change to an existing system needs to be analyzed.
3. A proposed system does not exist.
4. Options to improve or measure system performance need to be quantified.
5. Other quantitative analytic methods are computationally intractable.

Simulation allows one to ask "what if" questions and to derive new information from existing knowledge. The simulation activity, coupled with the evaluation of alternate designs and courses of action, can lead to a better understanding of system operations and management policies.

The widespread use of simulation as an effective manufacturing or system development decision aid has been thwarted by the requirement for extensive training and skill in the design and use of the simulation modeling technique. The frustration of simulation has been that domain experts who know how their systems operate, and who can describe in detail the system operation, have been unable to take advantage of simulation modeling. These experts have relied on experienced simulation analysts to design and develop the simulation model. This dependence on experienced analysts by the domain experts has made effective communication between these two parties imperative. The success of simulation activities depends on 1) how well the expert can describe the system to a simulation expert, 2) how well the simulation analyst can understand that system, and 3) the effectiveness of accurately translating systems descriptions and goals to a simulation model. IDEF2 is focused on improving the productivity of the simulation modeler by improving the ability of the simulation modeler to communicate model assumptions and designs to the domain expert.

Thus IDEF2 is a simulation model design tool that provides a language for representing models of the time varying behavior of resources in a manufacturing system, providing a framework for specification of math model based simulations. IDEF2 was designed to improve the process of design of representative simulation models that can be executed to predict what a system will do under specific conditions. The interested reader is referred to [Softech 81c] for additional details regarding this method.

Simulation Model Design from the IDEF2 Perspective

The IDEF2 method development was based on an extensive experience base with continuous, discrete, and network simulation languages design and the application of these languages to industrial problems. The primary designers of IDEF2 were A. Alan B. Pritsker, Robin J. Miner, and John F. Ippolito of Pritsker & Associates one of the leading industrial simulation companies in the United States. IDEF2 decomposes the design of a simulation model into the following four submodels.

1. Facility Submodel (used to specify the model of the agents of the system and environment).

2. Entity Flow Submodel (used to specify the model of the transformations that an entity undergoes).

3. Resource Disposition Submodel (specifies the logic of agent assignment to entity transform demands).

4. System Control Submodel (specifies the effect of transformation independent or system external events on the modeled system).

One of the goals of this submodel decomposition was the need to allow teams of analysts to easily partition the tasks associated with the construction of large models. It was also the intent that the submodel specifications could be reused as actual system specifications., e.g., use of the facilities submodel as a basis for other quantitative plant layout analysis, or the use of the resource disposition submodel and the system control submodel as the control logic specification for the shop floor control specifications. Finally, through the use of decomposition of the specification by behavioral partitioning, it was hoped that reuse of model specifications could be achieved.

IDEF2 Basic Concepts

IDEF2 is a graphical specification language with a computable interpretation. This means that simulation model designs are specified with a graphical syntax. However, they are complete enough to allow direct execution of the simulation model that they specify. This paper will not attempt to describe each element of the language in this brief summary, but rather illustrate each of the described submodels in the following figures.

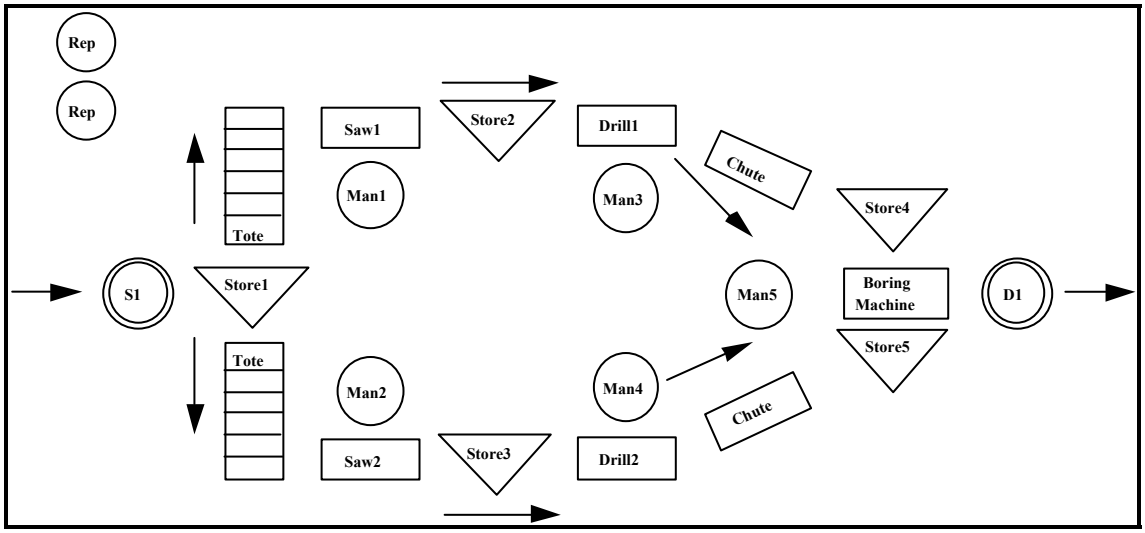

Figure 9. Facility Submodel used for Plant Layout

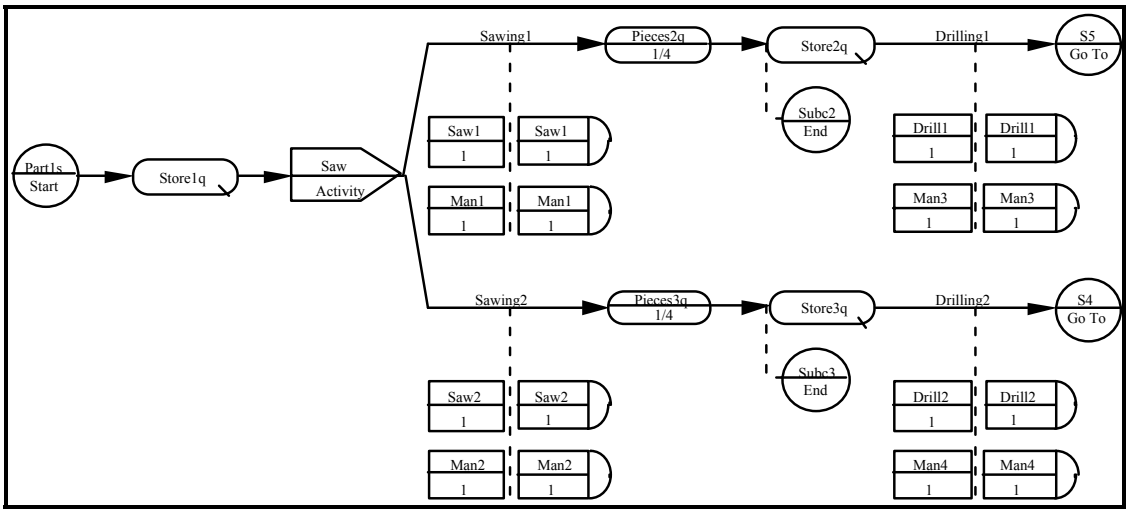

Figure 10. Example Entity Flow Network

Issues In IDEF2 Modeling

IDEF2 was widely used in CIM implementation initiatives and factory modernization efforts. A VAX-based simulation and decision support system developed by Pritsker & Associates was used to analyze these models. At this time, the IDEF2 method is relatively dormant. Many of the specification capabilities, and graphical innovations have been incorporated into commercially available simulation languages (e.g., MAP, SLAM, and SIMAN). IDEF2 demonstrates the ability to reduce the semantic gap between simulation model design and an executable simulation program. It represents an important advance for improving the productivity of simulation modelers, but does little to aid the non-simulation trained decision maker. This is similiar to a traditional CAD system that aids a product designer in quickly producing specifications for a mechanical part, but provides no support for the actual design decisions behind those specifications.

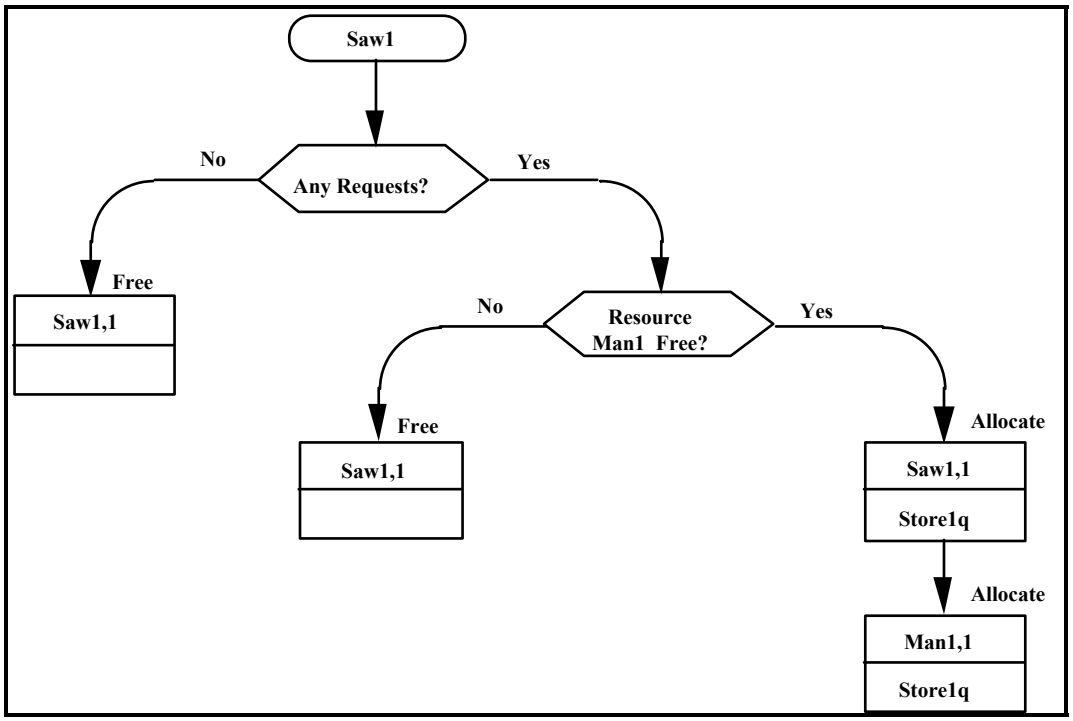

Figure 11. Example Resource Disposition Submodel

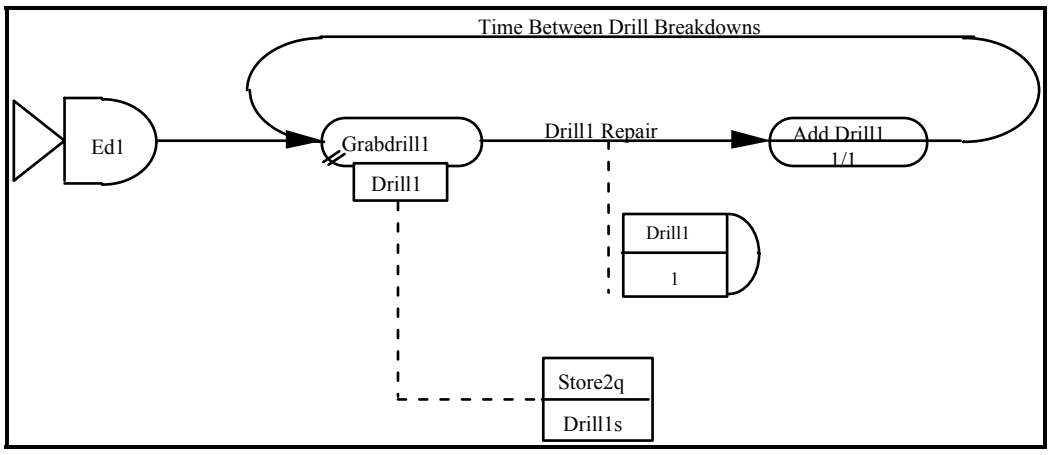

Figure 12. Example System Control Submodel

The IDEF3 method (described in the next section) is targeted at assisting the domain experts in recording their knowledge about process flow and object state transitions in their environment. Continuing research is being directed at the construction of knowledge based systems to automate the design of simulation models from these IDEF3 process descriptions and the analysis question to be answered. Such a system would provide support for the decision maker to input what is known about the system (a description of how his system works, and a question about his system that he wants answered). In these environments, IDEF2 can provide an intermediate representation of the generated model design for review by the simulation analyst. It is expected that IDEF2 will be updated to incorporate object-based (if not object-oriented) specification constructs that have recently emerged.

IDEF: The Next Generation

A method is a designed system. Unlike other CIM technology, a method is designed to execute on the human mind (more often on a multidisciplinary collection of minds). Like any other system, a method extended beyond its designed limits will fail. The goal of the IDEF developments has been to provide an interlocking framework of methods for the definition (or reverse engineering), design, development and maintenance of information integrated systems. The term *framework* [Zachman 87, IUG 90, Mayer 90] means a structured collection of methods, rules, procedures, and tools to support the development and evolution of systems. Experience during the first seven years of applying the IDEF

methods resulted in the identification of a number of additional method needs [Mayer 87] including the following.

1. Need to capture scenarios of logical or temporal sequences of events.
2. Need to design effective object-oriented applications and databases.
3. Need to capture reference descriptions of the objects described in the real-world realm.
4. Need to record CIM design decision rationale in order to achieve TQM on the CIM system itself.

These needs are being addressed by the next generation of IDEF methods, described in the following sections.

Process Flow & Object State Descriptions With IDEF3

One of the most common communication mechanisms to describe a situation or process is a story told as an ordered sequence of events or activities. IDEF3 is a scenario-driven process flow modeling method created specifically for these types of descriptive activities. IDEF3 is based upon the concept of direct capture of descriptions of the precedence and causality relations between situations and events in a form that is natural to domain experts in an environment. The goal of IDEF3 is to provide a structured method for expressing the domain expert's knowledge about how a particular system or organization works. An IDEF3 description can be used to provide the data for many purposes including the following.

1. To provide a systematic method for recording and analyzing the raw data that results from fact-finding interviews in a systems analysis project.
2. To determine the impact of an organization's information resource on the major operating scenarios of an enterprise.
3. To provide a mechanism for documenting the decision procedures affecting the states and life-cycle of critical shared data (particularly manufacturing, engineering, maintenance, and product definition data).
4. To define data configuration management and change control policy definition.

5. To support system design and design tradeoff analysis.

6. To provide powerful mechanisms to support the generation of simulation models.

7. To provide useful information for the creation of functional (IDEFØ) models.

8. To facilitate process mapping for the design of software to achieve real-time control by providing a mechanism for clearly defining the facts, decision points, and job classifications.

9. To provide an analyst with a method to clearly define the data needed to develop needs and requirements from a user viewpoint.

10. To collect and express the views of domain experts required for the development of expert systems.

The IDEF3 Process Description Capture Method is used by systems developers to capture domain expert knowledge about the "behavioral" aspects of an existing or proposed system. Process knowledge captured using IDEF3 is structured within the context of a scenario, making IDEF3 an intuitive knowledge acquisition device for describing a system. Unlike IDEFØ models which adopt a single perspective of the system and explicitly remove all temporal logic to promote generality and simplification, IDEF3 serves to structure different user descriptions of the temporal precedence and causality relationships associated with enterprise processes. The resulting IDEF3 descriptions provide a structured knowledge base from which analysis and design models can be constructed.

The development of IDEF3 was motivated by the need to distinguish between descriptions of what a system is supposed to do and mathematical idealizations, or models, that predict what a system will do. The IDEF2 Simulation Modeling Method and a host of other simulation languages (e.g., QGERT, SLAM, etc.) are targeted at satisfying the latter concern. Such languages represent the time-varying behavior of system resources and provide a framework for the specification of math model based simulations. IDEF3 addresses the former concern as a language for the organization and expression of different user views of the system. The organizational principles and concepts upon which IDEF3 is based come from pioneering work by 1) Dr. Shir Nijssen and 2) Dr. Jon Barwise [Barwise 83, Devlin 89]. Dr. Nijssen promoted the notion that

an information system is the embodiment of a discourse situation between agents in an organization. Dr. Barwise initiated an entirely new field of situation theory and situation semantics that promises to provide the theoretic basis for a new understanding of how any such discourse situation can come about and what flow of information can be supported by such a discourse situation. The interested reader is referred to [Mayer 89a, Menzel 90] for additional details regarding the theoretical background of this method.

Describing Systems From The IDEF3 Perspective

Two modeling modes exist within IDEF3: process flow description and object state transition description. Process flow descriptions are intended to capture knowledge of "how things work" in an organization. The object state transition description summarizes the allowable transitions an object may undergo throughout a particular process. Both the Process Flow Description and Object State Transition Description contain units of information that make up the description. These model entities form the basic units of an IDEF3 description. The resulting diagrams and text comprise a "description" as opposed to other IDEF methods that produce a "model". This distinction is an important one.

As discussed previously, a model is really an idealized system of objects, properties, and relations designed to imitate important aspects of a given real-world system. In a very real sense, models are themselves systems built around assumptions and simplifications of the real-world system presumed to hold true over the range of design situations to which the model will be applied to predict real-world behavior. A model must therefore be complete and internally consistent to ensure its usefulness.

Descriptions, however, are generally incomplete. For example, one might know something about a process or event outside of his specific area of expertise, but not know everything. Or facts could be omitted from a given description as irrelevant, or simply forgotten. Descriptions are simply recordings of facts and beliefs about the world around us that are assumed to be true, but incomplete. Model construction must therefore be preceded by the accumulation of descriptions provided by domain experts.

IDEF3 Basic Concepts

An IDEF3 Process Flow Description captures a network of relations between actions within the context of a specific scenario. The intent of this description is to show how

things work in a particular organization in the context of a particular problem solving (or recurring) situation. IDEF3 uses the "scenario" as the basic organizing structure for establishing the focus and boundary conditions for the process description. This feature of the method is motivated by the fact that humans tend to describe what they know in terms of an ordered sequence of activities they have experienced or observed within the context of a given scenario or situation. This natural tendency towards organizing thoughts and expressions within the context of a process description has motivated widespread use of the scenario as an informal framework for proposing alternative "external views" of possible system designs whose role will be to support the activities of the organization within the established context. Such development approaches have been referred to as "External Constraint Driven Design" approaches, and have been repeatedly demonstrated in practice as an effective mechanism for the design of new systems. Figure 13 presents an example IDEF3 Process Flow Diagram.

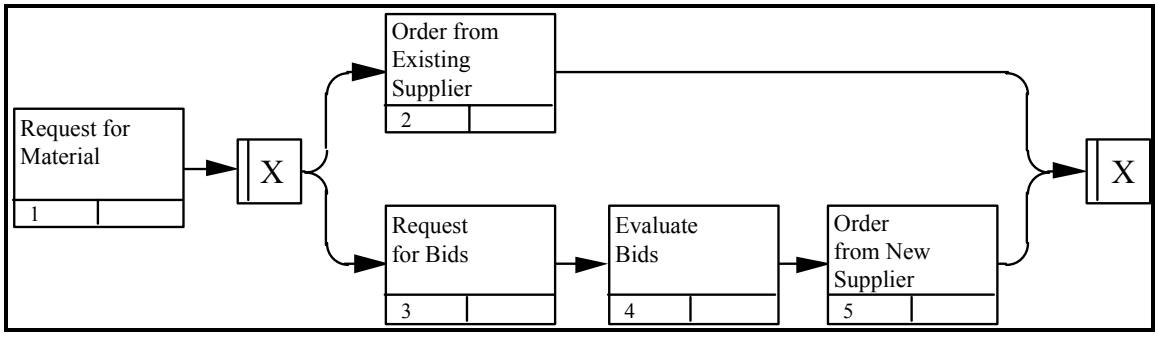

Figure 13. An Example Process Flow Diagram

An IDEF3 Process Flow Diagram consists of the following structures:

1. Units of Behavior (UOBs)
2. Junctions
3. Links
4. Referents
5. Elaborations

The development of an IDEF3 Process Flow Diagram will consist of the generation and manipulation of these descriptive entities.

The basic syntactic unit of IDEF3 graphical descriptions used within the context of a given scenario is the Unit of Behavior (UOB). This is the name given to what may be further classified as a function, activity, action, act, process, operation, event, scenario, decision, or procedure depending on its surrounding structure. Each UOB represents a specific view of the world in terms of a perceived state of affairs or state of change relative to the given scenario. Each UOB can have associated with it both "descriptions in terms of other UOBs", otherwise called decompositions, and a "description in terms of a set of participating objects and their relations", called elaborations. Note from Figure 13 that IDEFØ activities can be reused (and cross referenced with) IDEF3 UOBs.

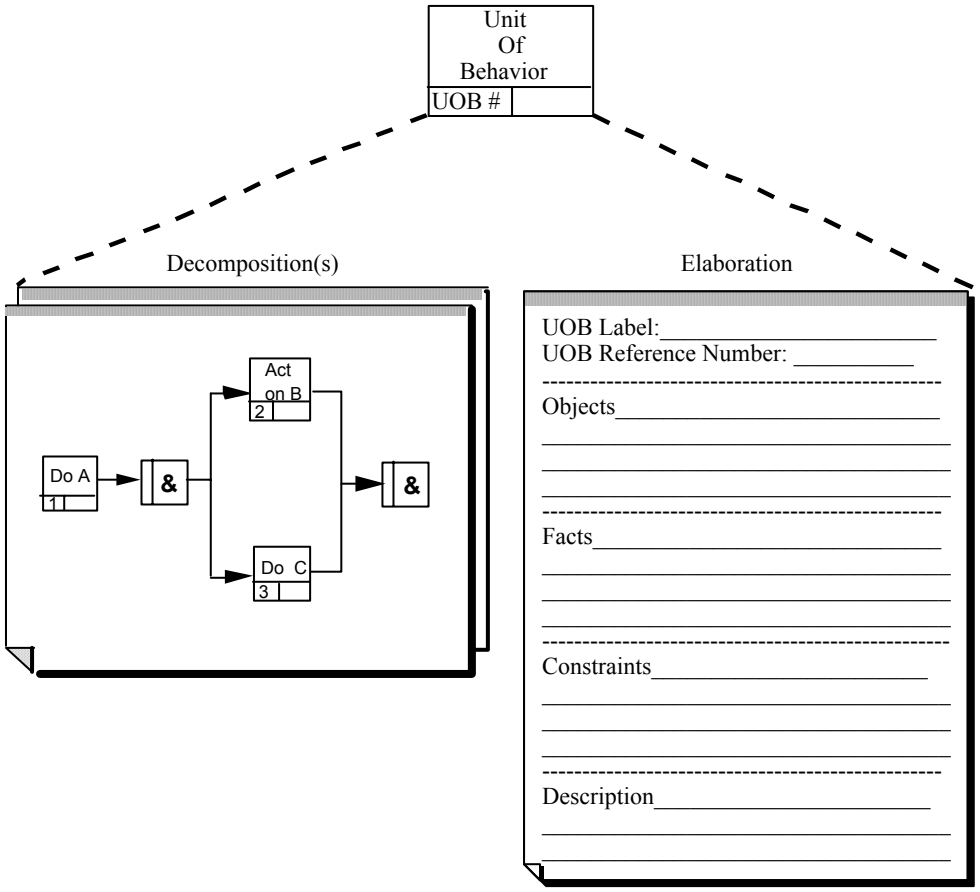

Figure 14. UOB with Its Decomposition and Elaboration

UOBs are connected to one other through the use of junctions and links. Junctions provide the semantic facilities for expressing synchronous and asynchronous behavior among a network of UOBs. Links are 1) temporal precedence, 2) object flow, and 3) relational. Relational links are provided to permit constraint capture not accommodated by the default semantics of the precedence and object flow links.

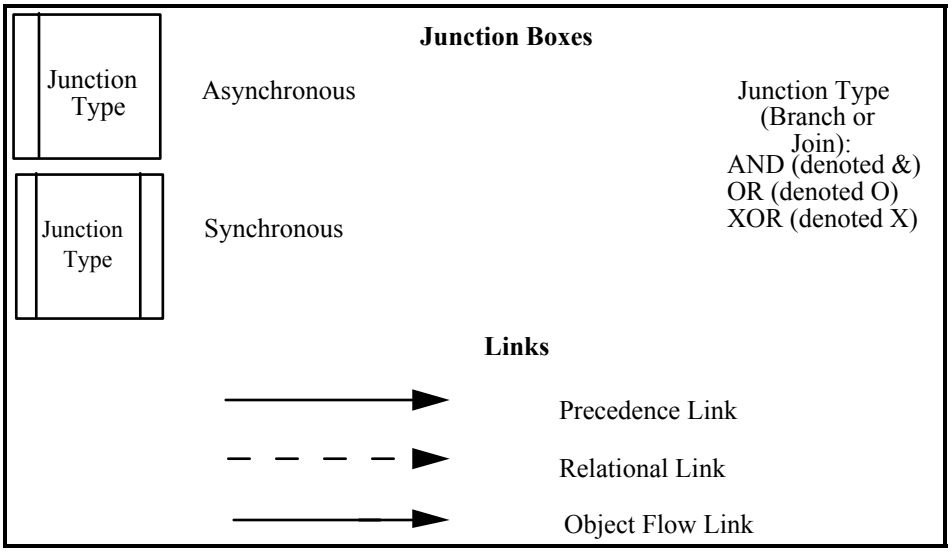

Figure 15. IDEF3 Junction Types, Link Types, and Their Meanings

An Object State Transition Diagram is used to capture an object-centered view of a process. This view cuts across the process diagrams and summarizes the allowable transitions of an object in the domain. The entities of an Object State Transition Description are the following.

1. Object States
2. State Transition Arcs

An object state is defined in terms of property values and constraints. The properties that are kept track of by the information systems must be defined as attributes in an IDEF1 model and cross referenced in the object state Transition Diagram. An Object State can also have Pre-transition and Post-transition constraints associated with it. These constraints specify the conditions that must be met: 1) before a transition can begin, or 2)

before a transition can be considered complete. The constraints are specified by a simple list of property/value pairs or by a constraint statement. The values of the attributes must match the specified values for the requirements to be met. The object state diagram also allow one to specify that an object must be processed through a particular process flow network before the transition from one state to the next is allowed. Figure 16 shows how an object state description would appear in an Object State Transition Network (OSTN) diagram. The solid circle represents the description of the actual state. Each object state has an associated elaboration. An OSD form is constructed for every object state represented in the OSTN diagram. In addition to enabling a detailed characterization of a state, the OSD form facilitates the specification of the requirements for all possible transitions in and out of the state as well as the requirements for the object to exist in a state. There are three types of requirements which are necessary to define a state. These are 1) *entry conditions* (for an object state) that must exist for the object to transition into a state; 2) *exit conditions* (for an object state) that must exist for the object to transition out of a state, and 3) *state description* that exist while an object is in a state. These conditions are expressed as attribute-value pairs and/or constraints.

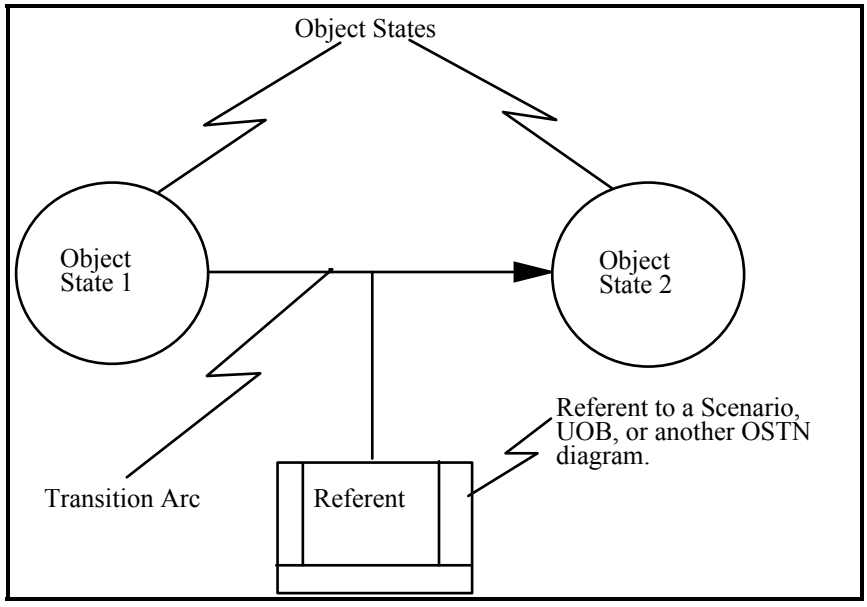

Figure 16. IDEF3 Object State Description

Guidelines When Using IDEF3

IDEF3 is designed to provide a medium for capturing the raw description of facts known by domain experts about how their system works. However, it does provide a rich variety of mechanisms for organizing and conveying those facts. Overly complex IDEF3 diagrams can result from combining many scenarios and viewpoints into a single diagram. A heuristic to constructing IDEF3 diagrams is if any of the displayed UOBs are not visible from all of the views associated with a scenario, it should probably be in a separate UOB. Also, beware of the tendency to "fill in the gaps" in the data collected. As a description capture method, IDEF3 was designed to be tolerant of "partial" and even inconsistent descriptions. In the analysis of organizations and the information systems that support those organizations, it is often just those areas of inconsistency or incompleteness that are the root of a particular problem. If the description hides these areas either by omission or by intention, the problems may go unnoticed giving rise to the oft heard comment, "These models don't indicate anything is wrong." Of course, they don't, since the modeler has polished over all of the rough areas which are exactly the areas of interest!

Object-oriented Design with IDEF4

Like IDEF1X, IDEF4 (Object-oriented Design Method) is intended to be used as a design method for automated systems implementation. IDEF4, however, targets the use of object-oriented technology, rather than relational technology, for the target implementation. The emergence of object-oriented philosophy and development practice have demonstrated the ability to produce code that exhibits desirable life-cycle qualities such as modularity, maintainability, and reusability. Likewise, the object-oriented programming paradigm has demonstrated major advancements in the ease with which software code can be created, enabling more people to successfully produce code. Paradoxically, this ease with which code can be produced also makes it easier to create software that is of poor design. Poor designs result in systems that are not modular, are difficult to maintain, and whose implementations are far more difficult to reuse. The goal of IDEF4 [Edwards 89] is to assist in the correct application of the Object-oriented Programming (OOP) paradigm to ensure consistent benefits from that technology. Before describing IDEF4, it is useful to review the background and philosophy of OOP.

Amid rising costs of serving the information needs of modern manufacturing corporations, one of the emerging technologies is OOP. Whether measured by cost, headcount, weight, or volume, information is a major product of every world-class manufacturing corporation. The need for reductions in information production costs has led to intense interest in a technology that started for user interfaces to CAD systems in the artificial intelligence (AI) labs at the Massachusetts Institute of Technology, was used by simulation groups for many years, again became the purview of the AI community as a possible knowledge representation paradigm, and emerged as a major software productivity paradigm. The impact and benefits of this paradigm in the engineering and manufacturing domain will depend upon how well it is actually understood.

One of the current problems plaguing OOX is the nature of the X. In the scramble to acquire the technology and the mad dash to supply technology which vendors don't have, there has been the tendency to redefine object-oriented (reminiscent of the *relational*, *three schema*, *rule based*, and other buzz words of the not so distant past). Thus, there are the terms *object-based* and *object-centered* as well as object-oriented sorts. Even Ada and IMS are presented as being "inherently" object-oriented, which, of course, might cause even the mildest skeptic to wonder what in the world is going on!

Presumably the intended goals of object-orientation was attaining goals of reusability, modifiability, reliability, maintainability, reduction of redundant code, and increased human ability to comprehend and implement more complex systems. An important feature of object-orientation in a programming language is its higher levels of abstraction. It is worthwhile to recount how the abstractions contribute to these goals.

Reusability is attained by two principal means: 1) through the encapsulation of the sets of routines operating on a data type (object class type) as a software module constituting the definition of that data type; and 2) through the concept of protocols, or declarative specifications of the purpose and effects of generic routines and their intended uses.

Modifiability and maintainability is enhanced through the same means as reusability. In addition, inheritance will make it possible to make alterations in a program simply by adding inheritance links, in cases where such alterations would require extensive rewriting of code (or design) in non-object-oriented programming languages (or design methods).

Reduction of redundant code is achieved again through the use of inheritance, which provides the effect of extensive copying of code (as in a macro definition) without any actual copying. Reducing redundancy also leads to increased reliability; redundant copies of code can easily lead to inconsistencies in much the same way redundancies in a database can.

Object-orientation is a conceptual power tool because it is so closely linked to taxonomic reasoning, which is fundamental to the human way of thinking about the ontology of problem domains. Though it must be emphasized that an object-class hierarchy is not the same as a taxonomy, the two are similar enough for object-orientation to significantly reduce the division between problem conceptualization and program coding.

The term *object* in the original computer language vernacular refers to a programming concept that supports creating and manipulating program objects consisting of some local state data (which represents relevant local state information) and some programmed behavior. A key phrase is *relevant local state information*. If one examines a large body of actual object-oriented programs, typically the classes of objects represent some information concept (possibly about a real-world object but generally not). Thus, the term *object* suffers from the phenomenon of use of a common term in a specialized manner. That is, the use of *object* in *object-oriented* in general does not refer to the physical object with the same name. Nor do object-class hierarchies in object-oriented designs normally correspond to the taxonomy hierarchy associated with the physical objects with the same names. Thus, *object-oriented programming* objects normally correspond to data things which may or may not correspond to or behave like the real thing! However, there are emerging sister paradigms to the OOP paradigm which attempt to move the conceptual power tool characteristics of the OOP up into the design, requirements, and even problem analysis stages of system development.

The following definitions are of some assistance in understanding the levels of object-orientation technology:

1. Object-based technology provides methodological analysis and conceptualization support based on analogy to real-world physical or conceptual objects. Object-based technology runs on the human brain.

2. Object-centered technology provides programming objects with local state storage, protocol, and method definition. However, the programmer must usually build his own basic inheritance structures for the object state and protocol methods.

3. Object-oriented technology provides full language and machine support (including type checking and multiple inheritance support) for the object state storage, protocol, and method definition and manipulation.

4. Persistent object-oriented technology extends the half-life of the objects providing disk resident support for storage of object-oriented structure definitions and instances.

Luckily, hardware and operating systems are advancing to the state where programmers no longer "need" to be concerned with low level details such as memory allocation. OOP based systems can raise the level of abstraction to the point where applications programmers can express the solution to problems in a clear maintainable fashion while letting the systems programmers worry about the details. This should allow widespread access to the benefits described above. As more programmers are allowed to adopt the object-oriented languages, the emerging technology of object-oriented databases (OODB) can be used to provide persistent objects, and to allow access to objects by other systems over networks. It is postulated that because of the more visible reference semantics of the OOP paradigm (i.e., it is more obvious what the data represents and how it is going to be used), system performance can improve 100 to 500 percent with OODB over relational systems.

Designing Systems From The IDEF4 Perspective

Whereas traditional methodologies, such as structure charts, data flow diagrams, etc. and data design models (hierarchical, relational, and network) have supported the development philosophy and practice for conventional systems development, IDEF4 seeks to provide the necessary facilities to support the object-oriented design decision making process. Specifically, the two primary design goals of IDEF4 are to 1) provide support for creating object-oriented designs whose implementations will exhibit desirable life-cycle qualities and reduce total implementation development time, and 2) make it easy to evaluate object-oriented code to determine whether or not the delivered product both conforms to the design and exhibits the desired life-cycle qualities.

Just as Structured Design (e.g., Structure Charts, Data Flow Diagrams) facilitated higher quality and productivity in the era of functional programming, the IDEF4 method is targeted at achieving similar advancement for object-oriented programming. Whereas the functional emphasis of structured design is paralleled by the functional emphasis of programming languages like Pascal, C, Modula-2, and Ada, the object-oriented emphasis of IDEF4 design is intended to serve the object-oriented emphasis of languages such as Smalltalk, CLOS, C++, and Eiffel. Thus, IDEF4 is intended as a mechanism for the *Design System* activity (See Figure 2).

IDEF4 maintains information about an object-oriented design in a manner which will preserve as many of the concept and notational advances made in previous efforts. Such consistency with the previous work should assist the practicing software engineer in learning and using effectively the IDEF4 modeling techniques. However, IDEF4 attempts to encapsulate the best practice experience of designing for object-oriented databases and programming environments. As such, IDEF4 is focused on the identification, manipulation, display and analysis of the following.

1. Object definitions including instance variables, class variables, temporary variables, and data / object types of the above variable values.

2. Object structures including the inheritance hierarchy or lattice structure and an individual objects visibility relative to other objects.

3. Individual object behavior including instantiation methods and constraints (pre-, post-, and during conditions), deletion methods and constraints, and an abstract description of the behavior

4. The protocol of the system of methods including location of the methods in the object inheritance lattice, type, number and ordering of the arguments, abstract description of the behavior of each protocol item, and specialization of the abstract behavior for the individual object classes.

As a design methodology IDEF4 was structured to expose the components of an object-oriented system design which are important for a design team to manage during the design phase of a system development process. The primary elements of an object-oriented design is the state maintenance (defined in the class structure) and the behavior sharing (described in the methods and inheritance structure). On the other hand, since

IDEF4 is used to design for implementation, there may be times when a record, structure, list, string, or some other common data structure is better suited than an object. If a common data structure is used, then IDEF4 provides a means by which that type can be specified along with the class and method specifications.

IDEF4 employs a unique organization structure to ensure that design models do not become cumbersome and difficult to use with increasingly larger projects. This is accomplished by dividing the IDEF4 model into a variety of submodels, diagrams, and data sheets. In other words, IDEF4 divides the object-oriented design activity into discrete, manageable chunks, with each subactivity supported by a graphical method highlighting the design decisions that must be made along with their impact on other perspectives of the design. No single diagram shows all the information contained in the IDEF4 design model. This limits complexity and allows rapid inspection for the desired information. Carefully designed overlap among diagram types ensures consistency among the different submodels. IDEF4 is a graphically oriented methodology for the object-oriented design of computer systems. It provides the necessary facilities to support the object-oriented design decision making process. Conceptually, an IDEF4 design model consists of two submodels: the Class Submodel and the Method Submodel. These two submodels are linked through the Dispatch Mapping, and combined capture all the information represented in a design model. However, due to the size of the Class and Method Submodels, the designer never sees these structures. Instead, the designer makes use of a collection of smaller diagrams that effectively capture the information represented in the Class and Method Submodels.

There are five diagram types used within IDEF4 to express the structure of the data object classes, inheritance relations, methods, and protocol of an object-oriented design. The following is a list of these diagrams and a brief description of their purpose.

1. Inheritance Diagrams specify the inheritance relations among classes.

2. Type Diagrams specify relations among classes defined through attributes of one class having instances of another class as values.

3. Protocol Diagrams specify the protocol for method invocation.

4. Method Taxonomy Diagrams classify method types by behavior similarity and links between class features and method types.

5. Client Diagrams illustrate *clients* and *suppliers* of methods.

There are also two specialized Data Sheets that accompany the diagrams.

1. Class Invariant Data Sheets (specify constraints which apply to every instance of a particular class of objects),

2. Contract Data Sheets (specify contracts that the methods in a method set must satisfy).

Understanding these diagrams requires a description of the basic concepts of IDEF4. These concepts are presented in the next section.

IDEF4 Basic Concepts

The concepts that exist within IDEF4 will be familiar to those with object-oriented experience. The same structures that exist in most object-oriented languages also exist in IDEF4. The most notable concepts in IDEF4 are:

1. Classes

2. Features

3. Inheritance Links

4. Type Links

5. Method Sets

The class is the basic syntactic unit in an IDEF4 design model. The characteristics of a class are represented by a collection of features. Each feature can be either public or private, where a public feature is accessible to all classes and a private feature is accessible only by the class and its subclasses. The power of the object-oriented paradigm comes through the inheritance of classes. When an inheritance relationship is specified, all features of the parent class (superclass) are passed on to the child class (subclass). When this occurs, the inherited features in the subclass maintain the same characteristics as in the superclass unless they are explicitly redefined. This inheritance provides the ability to build complex class structures from simple classes. Figure 17 shows an example class inheritance diagram and the effect of both single and multiple inheritance on a simple set of graphics objects.

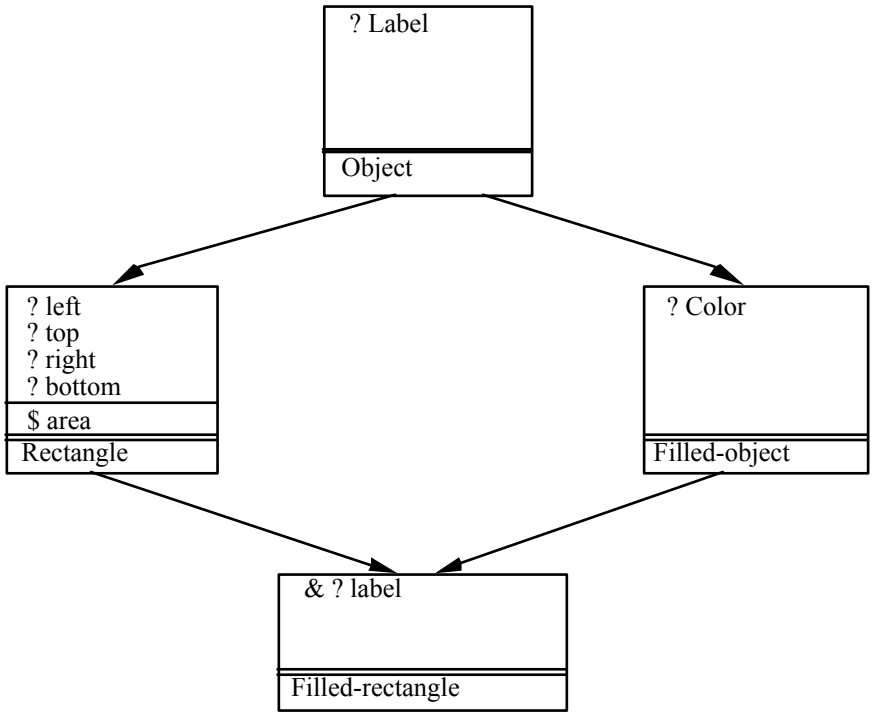

Figure 17. A Class Inheritance Diagram in IDEF4

Figure 18 shows how a class would appear in an IDEF4 Class Inheritance Diagram. The class is represented by a square-cornered box with the name of the class listed below the double line at the bottom of the box. IDEF4 requires that the first letter of the class name be capitalized. The features of the class are also displayed in the class box with private features displayed below the export line and public features displayed above the export line. The feature symbols provide additional information about the role that the particular feature plays.

For each class defined, the designer will attach a Class Invariant Data Sheet. This sheet is used to provide additional information about the objects in a class. The information represented in this data sheet must be true for all objects in the class at all times. An interesting feature of the Class Invariant Data Sheet is that subclasses also inherit the Data Sheet constraints of their superior.

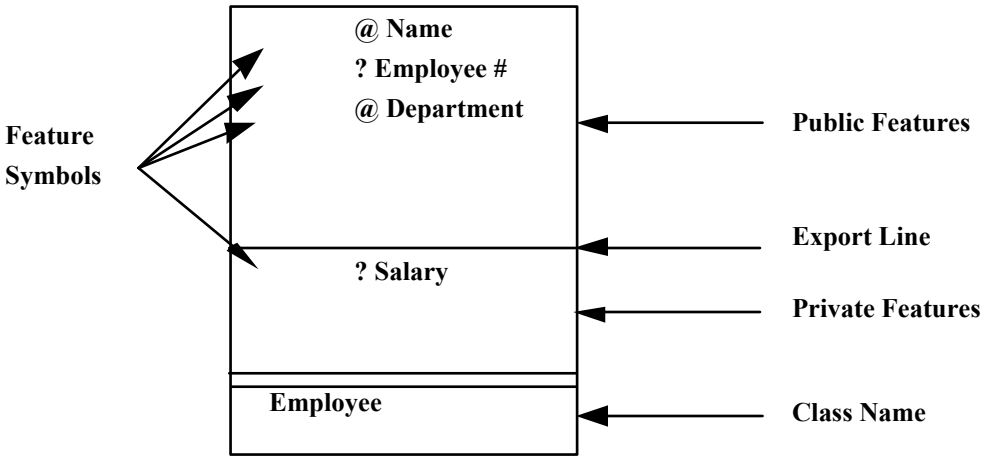

Figure 18. A Class Box in IDEF4

A feature is the named representation of a particular characteristic of a class. The features are used to capture the behavior of instances of a particular class. When the designer defines a feature, the type of the feature must be specified. It is important to distinguish between the type of the feature and the type of the value of the feature. Feature value type is concerned with the legal values that a feature may take or return. Feature type is concerned with the role that a feature will play within the context of a class. A feature can be only one of six types in an IDEF4 design model:

1. Non-Specific Feature (used as a place-holder early in the design process)
2. Routine (used to specify a feature that is to be implemented as a program)
3. Attribute (used to specify a feature that will hold or calculate a value)
4. Function (feature with characteristics of both a routine and an attribute)
5. Procedure (routine that is executed only for its side effect, doesn't return a value)
6. Slot (attribute that "holds" a value)

An Inheritance Link simply defines a parent/child relationship between two classes. When an inheritance link is specified, all characteristics of the parent class are inherited in the child class. Also, the inherited features will exhibit the same behavior in the child class as in the parent class unless the features are redefined using the auxiliary feature symbols.

A class can be considered to be a data type, and traditional programming data types can be considered to be classes. Since the features of classes that are classified as Attributes, or more specifically Slots or Routines, take on values, it would be useful to indicate the type of the value that the feature will take. These type declarations are made with Type Links. The Type Link specifies the feature being typed and the class that represents the legal values that the Attribute may take.

Figure 19 shows the four different Type Links supported in IDEF4. The first link simply says that the Attribute f of A returns a value of Type B. The second link is identical to the first except there is also a dual: while f of A returns a value of Type B, g of B returns a value of Type A. This dual link reduces the number of links that might appear in a Type Diagram and thus provides for a simpler diagram. The third link indicates that f of A returns a value that is constructed from B. This may be a list of instances of B or some other structure composed of instances of B. Finally, the fourth link provides a semi-dual for the third link type. While f of A returns a value constructed from instances of B, g of B returns a value of Type A. Figure 20 shows a type diagram for the graphical object example.

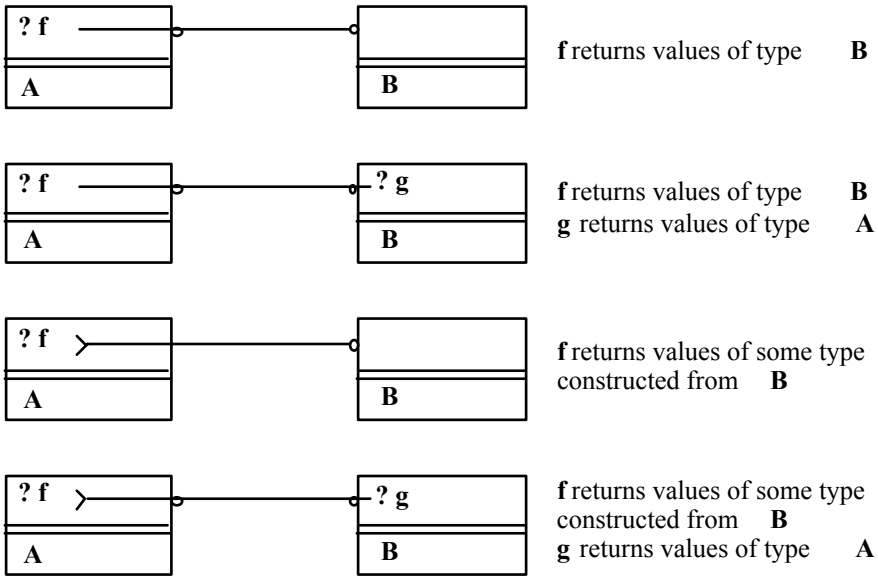

Figure 19. IDEF4 Link Types

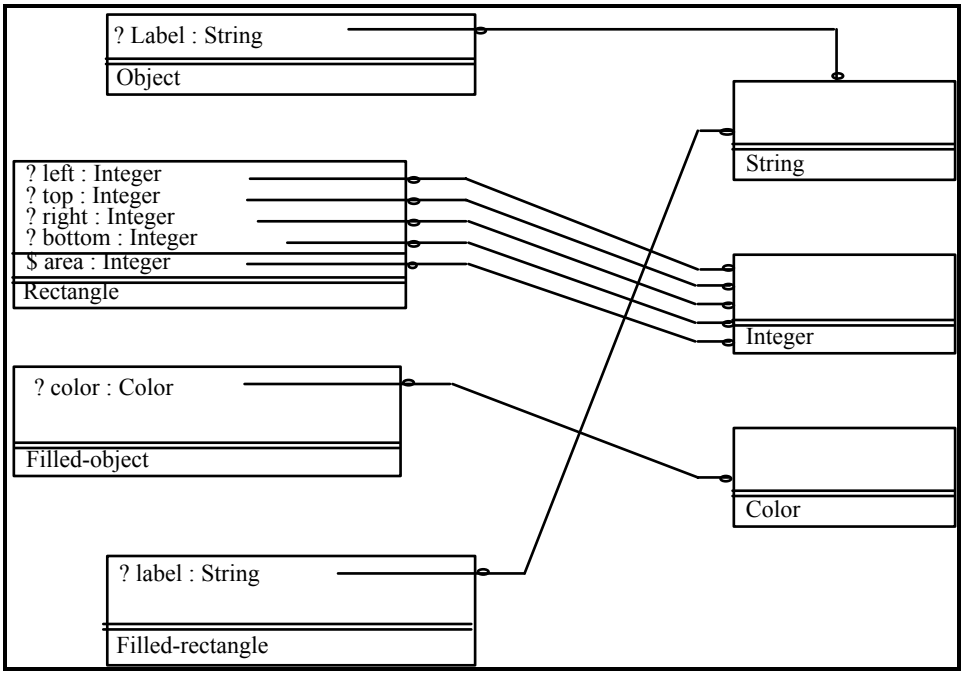

Figure 20. IDEF4 Type Diagram

IDEF4 does not represent individual methods. The reason for this is that a method could accept parameters that are instances of more than one class. As a result, the same method

must be defined for both classes. To alleviate this repetition and confusion, the information of these methods will be maintained in a method set. A feature of a class and that class will map to a method set. This mapping is referred to as Dispatch Mapping.

A method set is defined by its associated contract. In actuality, a method set is just a name for a contract data sheet. The contract data sheet maintains the declarative statements that define the intended effect of the methods in the method set. For a function, the contract would state the relationship between the function's argument list and the corresponding return values. For a procedure, the contract would have to specify how the method set changed the environment when passed an argument list and the current environment.

Guidelines When Using IDEF4

Many users of OOP have adopted the technology under the assumption that it eliminates the need for requirements analysis and design. This results in the same types of cultural problems with the use of IDEF4 as were experienced in the early days of traditional structured design (e.g., experienced OOP programmers don't see the reason, novice don't understand the concepts, and managers were hoping to eliminate a step in the life-cycle process.) Besides these cultural problems the IDEF4 method provides means for assisting in the focus on behavioral abstractions. However, new users typically use taxonomic abstractions in their formulation of the type diagrams. The natural consequence of this is that the method mapping between the type diagrams and the method set diagrams becomes very complex. This complexity is IDEF4's way of indicating to the designer that the class structure needs revision. Finally, IDEF4 was designed to assist in the design process. The assumption of IDEF4 was that it would be supporting an underactive design decision making process. However, it is quite common for the user of IDEF4 to treat the method simply as a means of documenting a design, often one that has already been coded.

Concept / Ontology Descriptions With IDEF5

In the context of information management, *ontology* is the task of extracting the nature and structure of a given engineering, manufacturing, business, or logistical domain and storing it in an usable representational medium. There is a pressing need for methods that can effectively capture what is known about the real-world and the relationships that exist between people, places, events, etc. The importance of capturing ontological information is especially crucial in the context of large systems. A large CIM project involves coordination of the resources of many different clusters of cooperative organizations. Each cluster makes its own contributions, and the overall success of the project depends on the degree of integration between those different clusters throughout the development process. A key to effective integration is a system ontology that can be accessed and modified across clusters that captures common features of the overall system relevant to the goals of disparate clusters. This *common* framework 1) promotes sharing of information arising from various sources within the system, 2) eases problems of information base maintenance, and 3) enhances the reusability of information once collected.

Rapid acquisition of reliable systems is perhaps the strongest motivation for ontology. Among the most significant problems in information management is the redundant effort expended capturing or re-creating information in systems that has already been recorded and developed. Rarely is this redundant effort expended purposefully. Rather, it is often the result of the inability of the development team to recognize the similarities or equivalencies between the situations. IDEF5 is targeted at the construction of reference models that can be used as a basis for both manual and automated identification of these similarities. IDEF5 can also be used as a precursor to enterprise information analysis. It provides a structure for recording and organizing the raw knowledge about physical and conceptual objects along with their natural associations. This fact base can then be used as the basis for a variety of analysis purposes (including determination of the information implications of these concepts). Finally, ontological analysis has been demonstrated to be an effective first step in the construction of robust knowledge based systems [Hobbs 87]. The current generation of CIM implementations will be taking advantage of knowledge based and expert systems technology. IDEF5 provides a method for the

initial knowledge acquisition for these systems. It also provides a representation of that knowledge that is independent of any particular implementation shell.

Ontologic inquiry has been the subject of extensive work within the information sciences. IDEF5 developments have drawn from such foundation work as Semantic Nets, Situation Theory, the NIAM Object Role Model, IDEF1, Set Theory, FOPL, the Modal Logics, etc. In so doing, IDEF5 attempts to fill a methodological gap not targeted by any other existing methodology. IDEF1 and IDEF1X capture primarily structural information, IDEFØ and IDEF3 various types of process information. Of course, since both structural information and process information involve objects in a system, there is the capacity for limited ontology representation within the existing methodologies. But, as noted below, there are several important kinds of ontological information that are not representable in those methodologies. Furthermore, those methodologies do not include techniques specifically designed for eliciting and capturing system ontologies. This suggests that there is a need for a separate method.

IDEF5 models the concepts and the conceptual relations for a domain. Conceptual modeling provides an abstract level of representation for describing a problem domain and / or system which closely reflects the human conceptualization of that domain including representation of real-world objects, attributes, and functions. An ontology represents the theory of what exists in a domain.

Describing Systems from the IDEF5 Perspective

An ontology can be thought of as a structure for representing knowledge about the world as perceived from different perspectives such that those perspectives can be related to one another. Ontologies are concerned with the identification and classification of concepts, objects, and associations together with the essential characteristics that identify the those kinds and associations.

Defining an ontology for a domain involves four major activities [Menzel 91]: 1) providing an inventory of the kinds of objects that exist within a given domain according to best sources of information regarding that domain (e.g., a domain expert); 2) for each kind of object, providing a description of the properties that are common to all and only instances of that kind; 3) characterizing the particular objects that in fact instantiate the

kinds within a system; and 4) providing an inventory of the associations that exist within a given domain between (and within) kinds of objects.

For example, consider the semiconductor manufacturing industry. The first two tasks might identify kinds of objects including wafers and reagents. Reagents may represent several subkinds including liquid reagents and etchants. Each if these kinds would have an associated set of necessary properties that its members contain.

The third and fourth tasks of ontology become more relevant in contexts where we want to be able to characterize specific individual objects, to speak specifically of them, their properties, and their associations. A basic distinction that is incorporated into IDEF5 is the distinction between essential and accidental properties. An essential property of an object is simply a property that the object could not possibly have lacked. An accidental property, by contrast, is a property that an object in fact has, but nonetheless might not have. The following section discusses how this feature in IDEF5 supports the rapid development of usable ontologies in the manufacturing domain which must deal with a rich combination of both natural and human designed objects.

IDEF5 Basic Concepts

The notion of "kind" (as distinct from class or type) is a central concept for IDEF5. It is important to recognize the distinction between the usual meaning of "kind" and what it represents in IDEF5. In naturally occurring systems it is often the case that all objects of the same kind have a distinguishing set of properties that must be maintained to remain a member of that kind. That is to say that the properties for membership are essential properties of the member. Thus, the usual notion of a kind is that of a collection of objects all of which share a common nature, i.e., a set of properties that belong essentially to all and only members of the kind. However, in the manufacturing systems it is frequently the case that objects must have a certain set of properties to become part of a kind but are not required to keep those properties to remain part of the kind. Consider semiconductor manufacturing as discussed above. A chemical has certain properties that identify it as an etchant, and all etchants have those properties. This is the traditional idea of a natural kind. Contrast this with the kind of object that a manufacturing "rework" item represents. A rework item might be any wafer that has more than three defects. Therefore, a wafer with four defects becomes a rework item.

However, after one or more of the defects on a wafer is repaired, it is still a rework item. In fact, it remains a rework item until it is reclassified by an inspector as an acceptable wafer or is discarded. This is an example of the "kinds" that typically arise in human designed systems. IDEF5 supports the identification of both kinds of kinds.

Put another way, the reason for the broader notion of a kind is that when an ontology is built for a certain human designed system, we are not just setting out to discover and classify the world as it is in itself, but rather to divide up and categorize the objects within the system in useful and informative ways. An ontology's categorization scheme is justified only insofar as it is useful to organizing, managing, and representing knowledge or information in the system so categorized. If objects of a certain kind K play a useful role in the system, that is all the justification one needs for admitting them into the system's ontology, irrespective of whether or not the defining properties of K are essential to its members (ask yourself, Is it necessary that your manager know how to manage?).

There is more to characterizing the objects in a system than listing their properties, though. For in the context of a given system it is equally important to detail the associations that objects in the system can, and do, bear to one another. Just as with properties, system-essential associations must be distinguished from system-accidental associations. This is partially because associations occur that way. It is also because the association may be a defining property of a kind (e.g., the marriage association and the kind "married"). A system-essential association relative to two (or more) kinds K_1, K_2 is an association that must hold whenever there are instances of K_1 and K_2. A system-accidental association relative to K_1 and K_2, by contrast, is one that need not hold between all possible instances of those kinds. Note that, just as defining properties of kinds neednot be essential to their instances, in the same way objects that stand in system-essential relations don't necessarily stand in those relations essentially; in human designed systems.

IDEF5 provides three diagram types for the visualization of an ontology. These diagrams are useful in both the construction and validation of the ontology.

Is-a diagrams are used in IDEF5 to show "is-a" relationships between kinds in an IDEF5 model. IDEF5 provides three types of is-a links: 1) generalization/specialization, 2)

AKO (a kind of), and 3) description subsumption. These link types are taken from [Brachman 83].

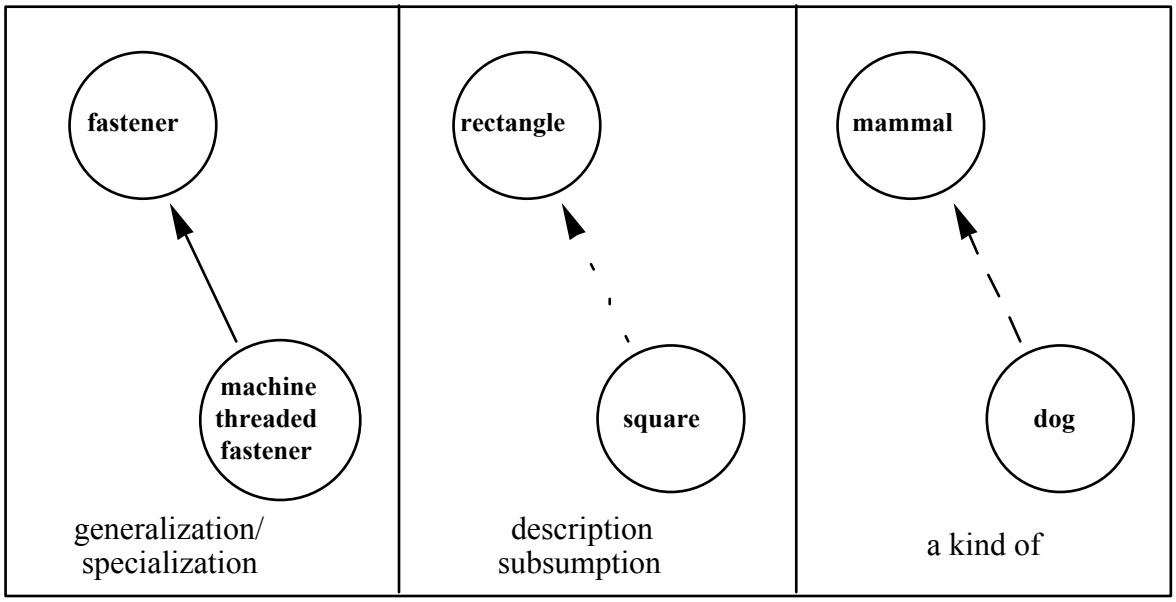

Figure 21. Is-a Link Types

Generalization/specialization links (also called superset/subset links) represent the specialization of a kind by another kind. For example, a **hex-headed bolt** kind is a specialization of a **fastener** kind for bolts with hex heads. AKO links are useful for representing natural kinds. For example, a **dog** kind is a kind of a **mammal** kind. Description subsumption links are useful for representing abstract kinds. The fact that "a square is a rectangle with four equal sides" is captured in a description subsumption link.

System kind diagrams are used in IDEF5 to show the kinds and relations that make up a system. Figure 22 is an example of a system kind diagram for a "Wafer Cutting System." Kinds in the system are represented by circles. System kinds in the system are represented by double circles. Lines between the circles represent relations, with the relation being from the tail of the arrow to the head of the arrow. System-essential relations are shown by double lines, and system-accidental relations are shown by single lines.

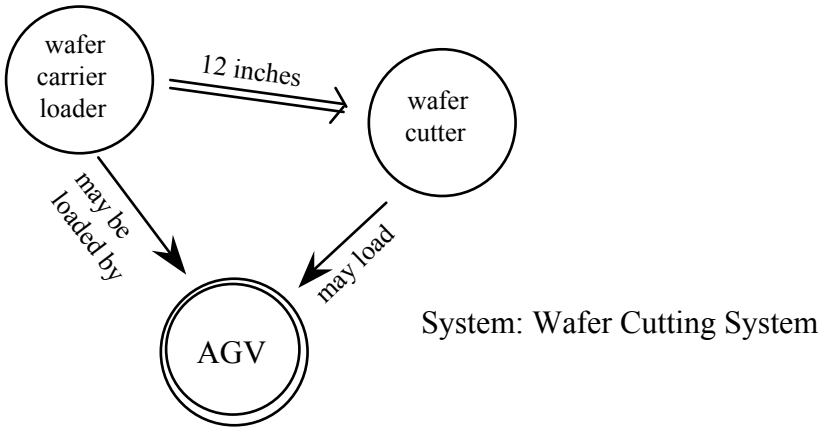

Figure 22. A System Kind Diagram

The third type of diagram in IDEF5 is the *relation type diagram*. This diagram shows the *axiomatization* of a relation in the model. To axiomatize a relation is to describe its meaning in terms of other relations in the system. This has traditionally been one of the most difficult parts of ontology development. The idea behind the relation type diagrams is to maximize reuse of a core set of relation definitions. Figure 23 shows a relation type diagram for the **has_sealant** relation.

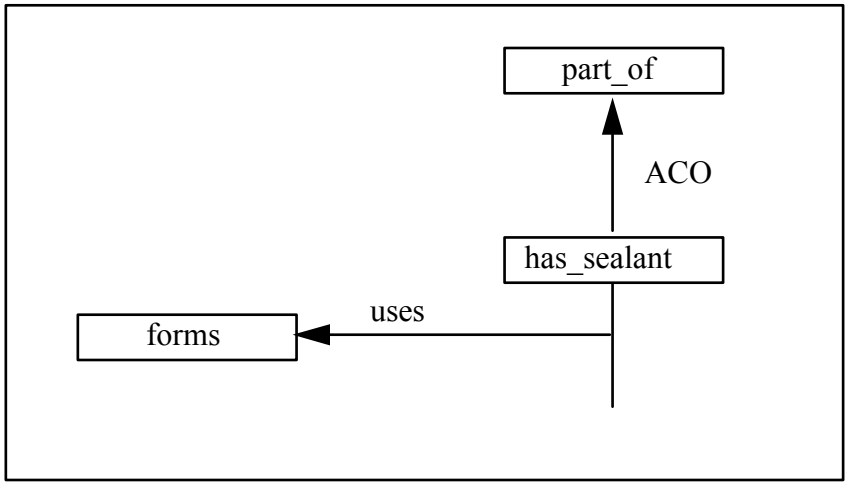

Figure 23. Relation Type Diagram

The **has_sealant** relation occurs between a **fastener** kind and a **sealant** kind. This relation reflects the fact that, in the automotive domain, there are fasteners that require sealants for some applications. The two relation type specifiers, as shown in the figure, are *ACO* (abides by the contract of) and *uses*. ACO links allows the ontology developer to reuse axioms that characterize one relation in the description of another. The uses link allows quick determination of the interrelationships between the relation characterizations.

Besides the diagram types IDEF5 includes a ten step process for the construction and application of the method. A more complete discussion of the theoretical foundations for the method can be found in [Menzel 91]. An in-depth description of the actual method can be found in [KBSI 91].

Things To Consider When Using IDEF5

IDEF5 provides the basic concepts for organization of the process of ontology development in a domain. Initial experience with this process indicates that it involves an iterative set of activities of: 1) fact data collection and analysis; 2) discovery of "proto-kinds" (initial guesses at the kinds); 3) refinement of the proto-kinds and their associations; 4) validation of the ontology against the original facts. One of the lessons learned is that steps 2 and 3 appear to be best performed by single individuals alone working on separate chunks of the data, where as steps 1 and 4 can easily be performed by a team of modelers.

Information System Design Rationale Capture With IDEF6

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 too have evolved from stand alone application oriented systems with relatively short lifetimes and limited scope towards 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, if not impossible 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 (like IDEF1X, IDEF2, and IDEF4) 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.

Design Rationale from the IDEF6 Perspective

The purpose of IDEF6 is to facilitate the acquisition, representation, and manipulation of the design rationale utilized in the development of enterprise level information systems. The term 'rationale' is interpreted as the "reason, justification, underlying motivation, or excuse" that moved the designer to select or adopt a particular strategy or system feature. More simply, 'rationale' is interpreted as the nature of the answer given to the question "Why is this design the way it is?" IDEF6 is intended to be a method with the concepts and language capabilities needed to represent information about the situations, relations, objects, states of affairs or courses of events that constitute system design rationale and associate that rationale with design specifications, models and documentation for the system. The scope of applicability of the technique component of IDEF6 is in the conceptual, preliminary and detailed design activities of the information system development process. To the extent that detailed design decisions for software systems are relegated to the coding phase the IDEF6 technique should be usable during the software construction process. Assumptions associated with the scope of IDEF6 include:

1. IDEF6 is targeted towards facilitation of the capture of design rationale for enterprise level information systems from the system level design to the detailed design of the implementation data structures, algorithms, user interface and processes.

2. It is unreasonable to expect designers to sit down at some point in time and "model" design rationale. Rationale must be captured at the source - at the point in time at which decisions are made.

3. People rarely write down design assumptions or rationale. To the extent possible it must be the case that IDEF6 be incorporated in a transparent manner into a wide variety of design methods (both formal and informal).

4. Design rationale is a small part of development decision rationale. That is, assume that design rationale will reference decisions on "what are important symptoms" and decisions defining what are the problems that give rise to those symptoms.

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 design commitments and propagate those commitments." IDEF6 characterizes both "types" of design rationale

and "mechanisms" for representation of these types. Types of design rationale identified for IDEF6 capture include:

1. Philosophy of a design including:
    A. Process descriptions of intended system operation.
    B. Design themes in terms of object or relation types.
2. Design limitations expressed as range restrictions on system parameters or environmental factors.
3. Factors considered in trade-off decisions.
4. Design goals expressed in terms of:
    A. Use or lack of use of particular components.
    B. Priorities on a problems requirements.
    C. Product life-cycle characteristics (e.g., disposable versus maintainable).
    D. 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 the carry-over strategy or the complexity of design rationale expression, the most common rationale given for a design is that it was the design that worked last year. Without making judgment on this situation, a minimum requirement for a design knowledge rationale capture capability is the ability to record historical precedence, as well as statements of beliefs and rationalizations for why a current design situation is identical to the one the previous design serviced. Another important rationale given for a design is just "it feels better," "it seems more balanced, symmetric." There is an important aesthetic side to software design.

Software design rationale includes expectations about how the design will evolve through the development process itself. For example, expectations about how the program structure will probably change - note such expectations do not appear to be as well defined as similar expectations we have seen in mechanical hardware design.

IDEF6 Basic Concepts

Thus, IDEF6 can be viewed as a structured technique for formulation of the types of design rationale statements (e.g., philosophy of the design, range restrictions or constraints, factors considered in trade-off decisions, design goals, etc.) It is also capable of supporting the formulation of such statements by simple reference to other life-cycle artifacts or objects. That is, if the reason for a particular design element is to satisfy a particular requirement constraint, then IDEF6 allows the statement of just this relationship with references to the requirement constraint (eliminating the need to reproduce the requirement constraint in the IDEF6 language). IDEF6 is still in its formulative stages. At this point of time, it takes the form of a language for the following.

1. Stating rationale.
2. Associating rationale statements with design elements.
3. Making and classifying "rationale" links between design elements and other life-cycle objects.

While IDEF6 could be applied in purely manual form, it is best suited for application in an automated environment that includes a life-cycle artifact repository (e.g., the IBM AD-cycle or DEC repositories or the Air Force Integrated Development Support Environment - IDSE). The IDEF6 language is based on an ontology of design rationale. That is, the language includes (as key words) a set of commonly used terms or phrases that express elements of rationale. An example of such a term and a phrase is the term "satisfies" and the phrase "is satisfied by" used in the following structures.

1. Design feature A *satisfies* the requirement B.
2. Requirement B *is satisfied by* design feature A.

Other terms/phrases that must be considered in an ontology of design rationale would include the following.

System

Subsystem

Component

Requires/Is Required By

Constrains/Is Constrained By

Bounds/Is Bounded By

Supports/Is Supported By

Creates/ Is Created By

Translates/Is Translated By

The IDEF6 language structure provides simple structured English-like sentence forms for employing these "rationale forming" constructs into statements associated with a design of a particular CIM system.

Issues In Design Rationale Capture

Since IDEF6 is still in the formulative stages, this paper describes factors associated with its application that have been discovered in the development process. For example, it is unreasonable to expect designers to introduce a separate step in the design process to document, or model, the assumptions or rationale upon which a given design decision is based. Therefore, much of the capture of this information must occur through background processes or interactive questioning initiated by a design support environment rather than the designer. This has influenced the IDEF6 method development in that it is assumed that the method will be used simultaneous with a number of different system design methods. Another important issue that has surfaced is that design rationale is just a part of the rationale motivating a development decision in the first place. Design rationale must therefore reference the symptoms motivating the system development decision and the probable causes giving rise to those symptoms.

Closing Remarks

Referring to Figure 2, this paper has presented some of the IDEF family of methods that have been used or are being developed to accomplish the analysis and design of a CIM system. Figure 24 shows a list of the current IDEF methods being developed. The reader may question the number of methods and the lack of one encompassing method capable of representing all that is needed to know about an existing or proposed system. Intuitively, it would be nice to have a single method representing all relevant perspectives of the system. This question can be answered by considering the purpose of models and descriptions from a slightly different perspective.

Generally speaking, the purpose of models and descriptions is to help make decisions. Each type of model or description focuses on a relatively narrow set of relationships and system characteristics comprising a particular viewpoint or perspective of the overall system. Analysis models, for example, are used to determine existing or anticipated design requirements. Design models serve to facilitate optimization of desirable design features for a restricted set of system requirements. Simulation models provide a perspective from which various measures and statistics associated with system performance can be generated to examine specific performance characteristics under a restricted set of operational conditions. Each model and the decisions generated through its construction carries with it a relative weighting towards overall system level decisions. Competing design decisions highlighted within and between model types eventually emerge, necessitating trade-offs.

IDEFØ Function Modeling
IDEF1 Information Modeling
IDEF1X Data Modeling
IDEF3 Process Description Capture
IDEF4 Object Oriented Design
IDEF5 Ontology Description Capture
IDEF6 Design Rationale Capture
IDEF8 User Interface Modeling
IDEF9 Scenario-driven IS Design
IDEF10 Implementation Architecture Modeling
IDEF11 Information Artifact Modeling
IDEF12 Organization Modeling
IDEF13 Three Schema Mapping Design
IDEF14 Network Design

Figure 24. Suite of IDEF Methods Including IICE Methods in Development

The goal of this process is an *optimal* design of the proposed system. Of course, designs or systems are considered optimal when evaluated against the current set of values, each of which is somehow manifested in the trade-off decisions made. This means that an optimal design does not necessarily, and most likely won't, exhibit all desirable life-cycle or performance characteristics.

As a result, models and descriptions focus on a limited set of system characteristics and explicitly ignore those characteristics not directly pertinent to the decisions at hand. Models and descriptions were never intended to represent every possible state or behavioral characteristic of a system. If such a goal were achievable, the exercise would itself constitute building the actual system, thus negating the benefits of modeling (e.g., low cost, rapid evaluation of anticipated performance, etc.). Having extended beyond the bounds of modeling into the realm of actual system construction, simulation becomes a statistical exercise rather than a design decision-making process.

The tendency to seek a single model to represent all relevant system life-cycle and behavioral characteristics, therefore, would necessitate skipping the design process altogether. Similarly, the search for a single method, or modeling language, to facilitate conceptualization, system analysis, and design continues to frustrate those making the attempt. Recognizably, the plethora of special purpose methods which typically provide few, if any, explicit mechanisms for integration with other methods, is equally frustrating. The IDEF family of methods is intended to strike a favorable balance between special purpose methods whose effective application is limited to specific problem types, and "super methods" which attempt to include all that could ever be needed. This balance is maintained within the IDEF family of methods by providing explicit mechanisms for integrating the results of individual method application.

Perhaps the most compelling argument for a family of methods is the ever-increasing need for methods that help manage complexity by dividing up the systems that must be analyzed, designed, and developed into discrete, manageable chunks. Methods are designed to embody knowledge of good practice for a given analysis, design, or fabrication activity. An appropriately designed method serves to raise the level of performance of the novice to a level comparable with that of an expert by focusing the modeler's attention on important decisions while masking out irrelevant information and unneeded complexity.

For the customer, floor plans and artists renderings are just as important as the final blueprints. It is therefore incumbent on the methods developer to constantly re-evaluate how well individual methods serve the needs of both the modeler and the customer. Practitioners must become sufficiently familiar with the basic theory behind the methods to ensure their appropriate selection and use.

Just as the original IDEF methods were targeted at managing the complexity associated with evolution towards large-scale integration in the manufacturing environment, new challenges will continue to emerge as those visions extend to integration across traditional boundaries as well. Large-scale integration between engineering, manufacturing, and support activities will be both exciting and challenging, particularly to the methods engineer. Their task will be to encapsulate the basic theory and body of experience associated with the analysis, design, and realization of tomorrow's integrated environments in easily usable forms

Bibliography

[ANSI 75] ANSI/X3/SPARC, Study Group on Data Base Management Systems: Interim Report, 75-02-08, In: ACM SIGMOD Newsletter, FDT, Vol. 7, No. 2, 1975.

[Barwise 83] Barwise, J. and Perry, J., *Situations and Attitudes*, The MIT Press, Cambridge, 1983.

[Cullinane 90] Cullinane, T., McCollom, N., Duran, P., and Thornhill, D. "The Human Elements of IDEF," Unpublished Paper, May 1990.

[Devlin 91] Devlin, K., *Logic and Information*, *Volume I: Situation Theory*, Cambridge University Press, 1991

[GE 85] General Electric, Integrated Information Support System (IISS). Volume 5. Common Data Model Subsystem. Part 4. Information Modeling Manual. IDEF1 Extended. DTIC-A181952, Dec, 1985.

[Feldmann 89] Feldmann, C.G., "Levels of Abstraction in IDEFØ Models," Unpublished Paper, October 10 1989.

[ISO 87] International Standards Organization, "Information Processing Systems - Concepts and Terminology for the Conceptual Schema and the Information Base," ISO/TR 9007, July 1, 1987.

[IUG 90] IDEF Users Group, "*IDEF - Framework, Draft Report*," IDEF-U.S.-0001, Version 1.2, Working Group 1 (Frameworks), Technical and Test Committee, IDEF - Users Group, May 22, 1990.

[KBSI 91a] Knowledge Based Systems Inc. (KBSI). The Nature of Ontological Knowledge: A Manufacturing Systems Perspective. *KBSI Technical Report Number KBSI-SBONT-91-TR-01-1291-01*, 1991.

[KBSI 91b] Knowledge Based Systems Inc. (KBSI). Formal Foundations for an Ontology Description Method. *KBSI Technical Report Number KBSI-SBONT-91-TR-01-1291-02*.

[KBSI 91c] Knowledge Based Systems Inc. (KBSI). Ontology Acquisition Method Requirements Document. *KBSI Technical Report Number KBSI-SBONT-91-TR-01-1291-03*.

[KBSI 91d] Knowledge Based Systems Inc. (KBSI). Ontology Capture Tool Requirements Document. *KBSI Technical Report Number KBSI-SBONT-91-TR-01-1291-04*.

[KBSI 91e] Knowledge Based Systems Inc. (KBSI). ***IDEF5** Method Report*. Prepared for U.S. Air Force Human Resources Laboratory, Contract No. F33615-C-90-0012.

[KBSI 91f] Knowledge Based Systems Inc. (KBSI). Reliable Object Based Architecture for Intelligent Controllers. *DARPA SBIR 91-050*. Contract No. DAAH01-91-C-R235.

[KBSI 91g] Knowledge Based Systems Inc. (KBSI). "Knowledge Based Information Model Integration," Final Technical Report, *NSF SBIR* . Award No. ISI-9060808, 1991.

[KBSI 92a] Knowledge Based Systems Inc. (KBSI), 1992. Ontology Capture Tool: Object-Oriented Design Document. *KBSI Technical Report Number KBSI-SBONT-91-TR-01-0292-01*.

[KBSI 92b] Knowledge Based Systems Inc. (KBSI), 1992. Knowledge-Based Automated Process Planning System with Assumption-Based Truth Maintenance System and Geometric Reasoning System. *DARPA SBIR 91-223*. Contract No. DAAH01-92-C-R066.

[KBSI 92c] Knowledge Based Systems, Inc, 1992, *IDEF3 Method Report*, Prepared for U.S. AL/HRG, Contract Number: F33615-90-C-0012.

[KBSI 92d] Knowledge Based Systems, Inc, 1992, *IDEF4 Method Report*, Prepared for U.S. AL/HRG, Contract Number: F33615-90-C-0012.

[Mayer 87] Mayer, R.J., et al., "Knowledge-Based Integrated Information Systems Development Methodologies Plan." Volume 2, DTIC-A195851, Dec. 1987.

[Mayer 90a] "IDEFØ Function Modeling: A Reconstruction of the Original Air Force Report," Mayer, R.J., editor, Knowledge Based Systems Inc. College Station, TX 1990a.

[Mayer 90b] "IDEF1 Information Modeling: A Reconstruction of the Original Air Force Report," Mayer, R.J., editor, Knowledge Based Systems Inc. College Station, TX 1990b.

[Mayer 90c] "IDEF1X Data Modeling: A Reconstruction of the Original Air Force Report," Mayer, R.J., editor, Knowledge Based Systems Inc. College Station, TX 1990c.

[Mayer 91a] Mayer, R.J., Menzel, C.P., and Mayer, P.S.D., "IDEF3: A Methodology for Process Description," Final Technical Report, Integrated Information Systems Evolution Environment Project, United States Air Force AL/HRGA, Wright-Patterson Air Force Base, OH, August, 1991.

[Mayer 91b] Mayer, R.J. , Edwards, D. A., Decker, L. P., and Ackley, K. A., "IDEF4 Technical Report," Integrated Information Systems Evolution Environment, United States Air Force AL/HRGA, Wright-Patterson Air Force Base, OH, July, 1991.

[Mayer 91c] Mayer, R.J., deWitte, P., Griffith, P., Menzel, C.P., "IDEF6 Concept Report," Integrated Information Systems Evolution Environment, United States Air Force AL/HRGA, Wright-Patterson Air Force Base, OH, July, 1991.

[Mayer 91d] Mayer, R.J., deWitte, P., *Framework Research Report*, Final Technical Report, Integrated Information Systems Evolution Environment, United States Air Force AL/HRGA, Wright-Patterson Air Force Base, OH, June 1991.

[Mayer 91e] Mayer, R.J., Painter, M., "IDEF Family of Methods," Technical Report, Knowledge Based Systems, Inc., College Station, TX. January, 1991.

[Mayer 91f] Mayer, R.J., et al., "Integrated *Development Support Environment (IDSE) Concepts and Standards*, Final Technical Report," Integrated Information Systems Evolution Environment Project, United States Air Force AL/HRGA, Wright-Patterson Air Force Base, OH, July, 1991.

[Mayer 91g] Mayer, R.J., Decker., L., "ISyCL Technical Report," KBSL-89-1002, Knowledge Based Systems Laboratory. AFHRL, Wright-Patterson Air Force Base, OH, 1991.

[Menzel 90] Menzel, C.P., Mayer, R.J., and Edwards, D., "IDEF3 Process Descriptions and Their Semantics," Kuziak, A., and Dagli, C., eds. *Knowledge Base Systems in Design and Manufacturing*, Chapman Publishing, 1990.

[Menzel 90] Knowledge Based Systems Laboratory. *IDEF3 Formalization Report.* Integrated Information Systems Evolution Environment, United States Air Force AL/HRGA, Wright-Patterson Air Force Base, OH, 1990.

[Menzel 91] Menzel, C.P., and Mayer, R.J., "IDEF5 Concept Report," Final Technical Report, Integrated Information Systems Evolution Environment, United States Air Force AL/HRGA, Wright-Patterson Air Force Base, OH, July, 1991.

[Painter 90] Painter, M.P., "Modeling with an IDEF Perspective: Some Practical Insights" Proceedings, SME Autofact 90, Detroit MI, 1990.

[Ross 85] Ross, D.T., "SADT Today: A Retrospective On An Idea". *IEEE Computer Magazine*, *1985 Special Issue on Requirements Engineering*, June 1985.

[SEM 83] "Analysis of IDEF Method Application in Industrial Practice", Interim Technical Report, Systems Engineering Methodology Program, Hughes Aircraft Corporation.

[Soley 90] Soley, R.M. (ed.), "Object Management Architecture Guide," Object Management Group, Inc.

[Softech 81a] Softech Inc., "Integrated Computer-Aided Manufacturing (ICAM) Architecture Part II, Volume IV, Function Modeling Manual (IDEFØ)," DTIC-B062457, June 1981a.

[Softech 81b] Softech Inc., "Integrated Computer-Aided Manufacturing (ICAM) Architecture Part II. Volume V. Information Modeling Manual (IDEF1)," DTIC-B062458, June 1981b.

[Softech 81c] Softech Inc., "Integrated Computer-Aided Manufacturing (ICAM) Architecture Part II. Volume VI. Dynamics Modeling Manual (IDEF2)," DTIC-B062458, June 1981b.

[Williamson 90] Williamson, W.R. "Effective IDEFØ Modeling—Some Tricks of the Trade," Unpublished Report, May 1990.

[Zachman 87] Zachman, J., "A Framework for Information Systems Architecture", *IBM Systems Journal*, Vol. 26 No. 3, September, 1987, pp. 276-292.