COGNITIVE SKILLS IN MODELING AND SIMULATION

Volume II

A Dissertation
by
RICHARD J. MAYER

Submitted to the Graduate College of
Texas A&M University
in partial fulfillment of the requirement for the degree of
DOCTOR OF PHILOSOPHY

December 1988

Major Subject: Industrial Engineering

COGNITIVE SKILLS IN MODELING AND SIMULATION

Volume II

A Dissertation

by

RICHARD J. MAYER

Approved as to style and content by:

Don T. Phillips
(Chair of Committee)

Leland Blank

Leland T. Blank
(Member)

Donald K Friesen

Donald K. Friesen
(Member)

G Kemble Bennett

G. Kemble Bennett
(Head of Department)

Peter J. Sharpe
(Member)

Guy H. Bailey

Guy H. Bailey
(Member)

December 1988

TABLE OF CONTENTS

Page

Volume I

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

Page

Volume II

LIST OF FIGURES

Figure Page

Volume I

LIST OF FIGURES (Continued)

Figure Page

Volume II

LIST OF FIGURES (Continued)

8. PROTOTYPE IMPLEMENTATIONS

This section summarizes the major engineering proof of concept prototypes which were constructed during the course of this research. The purpose of a proof of concept prototype implementation is to demonstrate the maturity of a particular design concept. The following prototypes are presented in this section:

1. An IDEF Based Methodology For Knowledge Acquisition (IDEF1/ES),
2. A Process Flow and Object State Modeling Method (IDEF3),
3. System Description Capture Environment Prototype (SDCE),
4. A Fact Collection Support Tool Prototype (FCT),
5. Model Development Support Environment Prototypes (MDSE),
6. An Extensible, Object-Based Simulation Engine (OBSIM),
7. Model Design Support from System Description Sketches (OBMODLER).

Each of these prototypes were constructed to test the feasibility of performing a particular task or providing a particular tool. They also serve as the experience base from which an assessment of full scale development resources can be estimated. Finally, the construction of such systems provides a reference point for discussion of architectural concepts, particularly in the areas of knowledge base construction.

8.1 IDEF1/ES and IDEF3: Methodologies For Knowledge Acquisition

The following subsections will describe the IDEF1/ES and the IDEF3 methods. The components of a method are displayed in Figure 8.1. Of these components, we will present the concepts and display syntax. We will provide an overview of the motivation behind the concepts but will not provide the formal syntax and semantics of each method. We will also provide some insight into the technique for application of each method. Since the IDEF1/ES is built directly from the IDEF1 base and since the IDEF3 heavily utilizes the IDEF1 concepts, for reference the

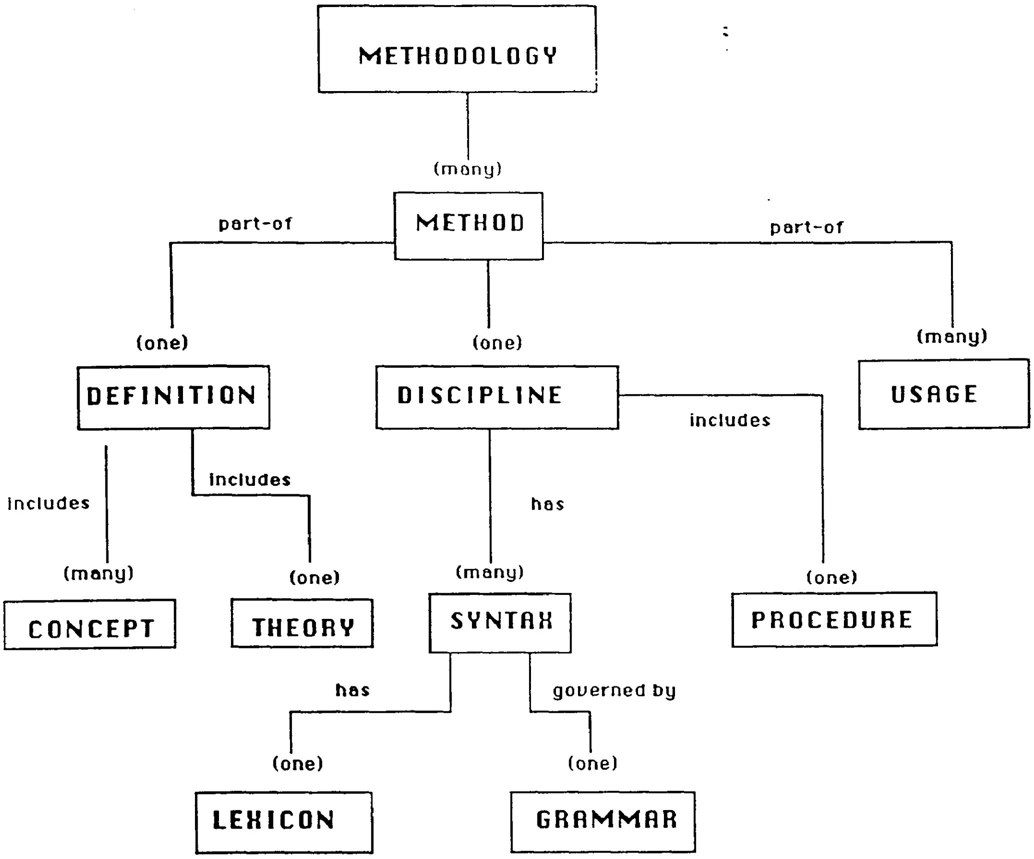


FIGURE 8.1: ELEMENTS OF A METHOD.

formal semantics of the original IDEF1 method are provided in the following subsection. [1]

The IDEF1/ES additions to the IDEF1 semantics can be summarized as follows:

1. Addition of the concept of Entity "Types" (allowing indexicality over subsets of properties in an entity binding),
2. Recognition of the individual named entity member (real world object image) as a necessary mechanism for assertional statements,
3. Provision for representation of the individual named real world object (type instance) as a necessary mechanism for assertional statements,
4. Allowance for the explicit representation of entity "Types," "Instances," "Classes," and "Members," indexed by definite (or indefinite) descriptions,
5. Introduction of the notion of event "Types," "Instances," "Classes" and "Members" with provision for both "Named" and "Description" references to each,
6. Expansion of the declarative constraint specification capabilities of IDEF1 to include:

   6.1. Uniqueness constraints on relations,

   6.2. Subsetting constraints on relations,

   6.3. Exclusion constraints on relations,

   6.4. Inclusively independent constraints,

   6.5. Exclusively dependent constraints,

[1] This semantics was the result of a cooperative effort between the author, Dr. Chris Menzel, and Timothy Ramey, the original developer of IDEF1. Many of the insights are theirs; I assume the responsibilities for the errors.

7. Procedural constraint specification, and the representation of procedural constraint collections,

8. Procedural, enumerative, and declarative cardinality constraints,

9. Timing and sequencing constraints,

10. Quantification over constraints,

11. Definition of rules for the kinds of relationships which can be established between the model element types.

The IDEF3 methodology was designed for the capture of scenario based process flow descriptions and the relation of those flows to enterprise object states. A need was identified in the manufacturing system description capture area for a method which would support the description of the timing, sequencing, and causality relationships between states of affairs and states of change. Expressing such relationships involves the identification of triggers (causality relations), initiation conditions, and completion conditions on the activities. The identification of these pieces of information requires specification of:

1. Timing constraints on individual activities and on groups of activities,

2. Sequencing constraints on groups of activities,

3. Attribute values and attribute value constraints.

The resulting method integrates information from the IDEF0, IDEF1, and IDEF1/ES methods.

8.1.1 IDEF1 Formalization

8.1.1.1 Lexicon

A *language* consists of two components: a *lexicon* and a *grammar*. The IDEF1 lexicon is a finite set with the following elements:

1. *Entity class names*: $A_1, A_2, \ldots, A_n$,

2. *Owned attribute class names*: $a_1, a_2, \ldots, a_m$,

3. Three types of *link names*:

   3.1. $L_1^\circ, L_2^\circ, \ldots, L_i^\circ$ ("weak" one-to-many),

   3.2. $L_1^\bullet, L_2^\bullet, \ldots, L_i^\bullet$ ("strong" one-to-many),

   3.3. $L_1^\rightarrow, L_2^\rightarrow, \ldots, L_i^\rightarrow$ (one-to-one),

4. Punctuation: [,], (,), ⟨, ⟩, and , (comma)

In the metalanguage we will use the following to discuss the syntax: 'ε' (perhaps with primes and subscripts) will range over entity class names. 'ω' will range over owned attribute class names. '$\lambda^\rightarrow$', '$\lambda^\bullet$', 'λ°' will range over the appropriate link names, and 'λ' will range over link names generally.

8.1.1.2 Grammar

1. If $\lambda_1, \ldots, \lambda_n$ are pairwise distinct link names and ω is an owned attribute class name, then $\omega \cdot \lambda_1 \cdot \ldots \cdot \lambda_n$ is an *inherited attribute class name.* We will use 'ι' to range over inherited attribute class names and 'α' to range over attribute class names generally. An inherited attribute class name $\iota = \omega \cdot \lambda_1 \cdot \ldots \cdot \lambda_n$ is *one-to-one* iff λ_1 is a one-to-one link name.

2. For any inherited attribute class name $\iota = \alpha \cdot \lambda$, let $\pi_1(\iota) = \alpha$ and $\pi_2(\iota) = \lambda$. For owned attribute classes ω, $\pi_1(\omega) = \pi_2(\omega) = \omega$.[2]

3. If $\alpha_1, \ldots, \alpha_n$ are pairwise distinct attribute class names such that either each α_i is a one-to-one inherited attribute class name or none is, then $(\alpha_1, \ldots, \alpha_n)$ is a *key class name.* 'κ' will be used to range over key class names.

4. For any key class name $\kappa = (\alpha_1, \ldots, \alpha_n)$, where $n \geq 1$, let $AN(\kappa) = \{\alpha_1, \ldots, \alpha_n\}$[3]

[2] $\pi_1(\iota)$ represents an attribute class in a key class of some parent entity class that is composed with some link to form an inherited attribute class in some child entity class.

[3] $\{\alpha_1, \ldots, \alpha_n\}$, that is, is the set of attribute class names between the parentheses in κ; it is *not* a metalingistic symbol for a construction in the grammar.

5. If ε is an entity class name and $\kappa_1, \ldots, \kappa_n$, $n \geq 1$, are key class names such that

   5.1. For no distinct $i, j \leq n$, $AN(\kappa_i) \subseteq AN(\kappa_j)$,

   5.2. $\mathbf{I} = \{\iota_1, \ldots, \iota_m\}$ is a set of inherited attribute class names such that for any $\lambda^{\rightarrow}$, if $\{\iota_i \mid 1 \leq i \leq m \text{ and } \pi_2(\iota) = \lambda\} \neq \emptyset$, then there is a $\kappa_j, 1 \leq j \leq n$, such that $\{\iota_i \mid 1 \leq i \leq m \text{ and } \pi_2(\iota) = \lambda\} = \kappa_j$, and for all $\iota_i, 1 \leq i \leq m$, there is exactly one such κ_j such that $\iota_i \in \kappa_j$, and

   5.3. $\mathbf{O} = \{\omega_1, \ldots, \omega_k\}$ is a set of owned attribute class names such that $\bigcup\{AN(\kappa_i) \mid i \leq n\} \subseteq \mathbf{I} \cup \mathbf{O}$, then

$$\varepsilon[\langle\kappa_1, \ldots, \kappa_n\rangle\langle\iota_1, \ldots, \iota_m\rangle\langle\omega_1, \ldots, \omega_k\rangle]$$

is an *entity class scheme.* 'σ' will range over entity class schemes.

6. For any entity class scheme σ in the above form, let $ECN(\sigma) = \varepsilon$, $KCN(\sigma) = \{\kappa_1, \ldots, \kappa_n\}$, $IAN(\sigma) = \{\iota_1, \ldots, \iota_m\}$, and $OAN(\sigma) = \{\omega_1, \ldots, \omega_k\}$.

7. For any entity class schemes σ, σ', we say that σ is *linked to σ' via λ* if for some $\iota \in IAN(\sigma)$, $\lambda = \pi_2(\iota)$ and for some $\kappa \in KCN(\sigma'), \pi_1(\iota) \in AN(\kappa)$.

8. Let $S \subseteq ECS$, let $S' = \{\sigma_1, \ldots, \sigma_n\} \subseteq S$, let $\mathbf{L} = \{\lambda_1, \ldots, \lambda_{n-1}\} \subseteq \{\pi_2(\iota) \mid \iota \in \bigcup\{IAN(\sigma) \mid \sigma \in S'\}$, and let $P = \{S, L\}$. Then,

   8.1. P is a *walk* (from σ_1 to σ_n) in S iff for all $i < n$, σ_i is linked to σ_{i+1} via λ_i or σ_{i+1} is linked to σ_i via λ_i.

   8.2. P is a *path* (from σ_i to σ_n) in S iff for all $i < n$, σ_i is linked to σ_{i+1} via λ_i.

   8.3. If P is a path, P is *increasing* iff for all $i < n$ there is a $\lambda^{\rightarrow}$ such that $\lambda_i = \lambda^{\rightarrow}$, and P is *decreasing* iff for all $i < n$ there is a $\lambda^{\bullet}$ such that $\lambda_i = \lambda^{\bullet}$.

8.4. If P is a walk, P is *increasing* iff $P' = \{S, L'\}$ is an increasing path, where L' is the result of replacing all the strong many-one link names in L with new one-to-one link names.

8.5. P is *cyclic* iff $\sigma_1 = \sigma_n$.

8.6. S is *connected* iff for all $\sigma, \sigma' \in S$, there is a walk from σ to σ' in S.

9. A set $S \subseteq ECS$ is a *prediagram* iff:

9.1. S is finite and connected,

9.2. for all distinct $\sigma, \sigma' \in S$, $ECN(\sigma) \neq ECN(\sigma')$, $OAN(\sigma) \cap OAN(\sigma') = \emptyset$, and $\{\pi_2(\iota) \mid \iota \in IAN(\sigma)\} \cap \{\pi_2(\iota) \mid \iota \in IAN(\sigma')\} = \emptyset$,[4]

9.3. for all $\sigma \in S$, and for all $\iota_1, \ldots, \iota_n \in IAN(\sigma)$ such that for all $i, j \leq n$, $\pi_2(\iota_i) = \pi_2(\iota_j)$, there is exactly one $\sigma' \in S$ and one $\kappa \in KCN(\sigma')$ such that $\{\pi_1(\iota_1), \ldots, \pi_1(\iota_n)\} \subseteq AN(\kappa)$.[5]

10. A prediagram S is an *IDEF1 diagram* iff:

[4] This requirement is simply that different entity class schemes cannot have the same entity class name, cannot share any of the same owned attribute class names, and their inherited attribute class names must differ at least in their rightmost link name; this amounts to the requirement that no more than one entity class scheme can be linked to any other via a given link name.

It should be noted that the first two parts of this requirement might appear to contradict a common modeling practice — inheritance is usually represented by writing the same attribute class name in two boxes. In fact, though, in a fully specified IDEF1 model, the complete "heritage" of an inherited attribute class is represented, links and all, essentially as is required explicitly here. It is recommended that the above mentioned practice be very carefully reconsidered as a part of rethinking the notion of inheritance as a whole as it is a potential cause (indeed an actual cause) of great confusion and lack of clarity.

[5] This is the requirement that all inherited attribute classes that are "inherited through" the same link "come from" the same key class in the entity class at the "front" of the link.

10.1. For all $\sigma, \sigma' \in S$ and for all $\lambda^{\rightarrow}$, if σ i s linked to σ' via $\lambda^{\rightarrow}$, then there is a $\kappa \in KCN(\sigma)$ such that for all $\alpha \in AN(\kappa)$, $\pi_2(\alpha) = \lambda^{\rightarrow}$, and there is a $\kappa' \in KCN(\sigma')$ such that for all α, $\pi_1(\alpha) \in AN(\kappa')$ iff $\alpha \in AN(\kappa)$;[6]

10.2. For all $\sigma, \sigma' \in S$, if σ is linked to σ' via a many-to-one link λ, then there is no $\kappa \in KCN(\sigma)$ such that for some $\kappa' \in KCN(\sigma'), \{\pi_2(\alpha) | \alpha \in AN(\kappa)\} \subseteq AN(\kappa')$;[7] and

10.3. No cyclic walk in S is increasing.

8.1.1.3 IDEF1 Formal Semantics

To provide formal interpretations for the well-formed (but as yet meaningless) expressions generated by the syntax, a mathematical structure is defined to interpret the elements of our chosen lexicon (other than punctuation). Specifically an **information structure** S is a 4-tuple $\langle \mathcal{E}, \mathcal{V}, \mathcal{L}, \mathcal{D} \rangle$ such that:

1. $\mathcal{E}$ is a finite set of pairwise disjoint finite sets.

[6] This requirement is especially emphasized in IDEF1. In English, it says that if one entity class scheme is linked to another via a one-to-one link name, then some key class name of the former must "replicate" a key class name of the latter, in the sense that the latter's key class name can be obtained by dropping the rightmost link name (the one that "causes" the link) from all the attribute class names in the relevant key class in the former. The semantic intuition is that if one entity class is linked to another via a one-to-one link function, then the attribute values that individuate the members of the latter class will suffice to individuate the members of the former, since no two of its members is mapped to the same member of the latter.

[7] In English, if an entity class name σ is linked to another σ' via a many-to-one link (of either sort), then there cannot be a key class name in σ that "replicates" (in the sense described above) all or part of some key class name in σ'. The guiding intuition here is that if one entity class E is linked to another E$'$ via a many-to-one link function, then at least two members a and b of the former get mapped to the same member c of the latter, in which case the set of values we get by applying the attributes in any key class of E$'$ to c will not suffice to distinguish a and b.

2. $\mathcal{L}$ is a function on $\mathcal{E} \times \mathcal{E}$ such that $\mathcal{L}(e,e')={}^{e}e'$ (i.e. the set of all functions from e into e');

3. $\mathcal{V}$ is a function on $\mathbf{N}$ whose range is a set of sets such that for all $e \in \mathcal{E}$, $i \in \mathbf{N}, e \cap \mathcal{V}(i) = \emptyset$;

4. $\mathcal{D}$ is a function on $\mathcal{E} \times \mathbf{N}$ such that $\mathcal{D}(e,i)={}^{e}(\mathcal{V}(i))$.

Intuitively, $\mathcal{E}$ represents the **meanings** of the entity class names, viz., the entity classes they denote. The purpose of the other objects is to associate two kinds of things with each element of $\mathcal{E}$:

1. A set of link functions which associate elements of the entity class with elements of other entity classes,

2. a set of owned attribute classes each of which maps every element of the entity class to a specific value in a given set of values.

The notion of an inherited attribute class is captured by composing link functions with owned attribute classes. For example, the meaning of the inherited attribute class DEPT-NAME within the EMPLOYEE entity class is the composition of the owned attribute class DEPT-NAME within the DEPARTMENT entity class with the link function WORKS-FOR that maps each employee to the department he or she works for. Key classes are then constructed out of the two types of attribute classes in the obvious way.

We will now make this more precise by means of a function *val* that maps elements of the lexicon into objects in our structure. Formally, we say that an interpretation $\mathcal{I}$ is a pair $\langle \mathcal{S}, val \rangle$, where $\mathcal{S}$ is an information structure and *val* is a function on LEX such that:

1. For all ε, $val(\varepsilon) \in \mathcal{E}$.

2. For all ω, $val(\omega) \in \bigcup\{\mathcal{D}(e,i) \mid e \in \mathcal{E}, i \in N\}$.

3. For all $\lambda^{\rightarrow}$, $val(\lambda^{\rightarrow}) \in \{l \in \mathcal{L} \mid l \text{ is injective (one-to-one)}\}$.

4. For all $\lambda^{\bullet}$, $val(\lambda^{\bullet}) \in \{l \in \mathcal{L} \mid l \text{ is surjective (onto)}\}$.

5. For all $\lambda^{\diamond}$, $val(\lambda^{\diamond}) \in \mathcal{L}$.

Given an interpretation $\mathcal{I} = \langle S, val \rangle$, we define val' recursively to be an extension of val such that for every $\iota \in IAN(\sigma), \sigma \in S, val'(\iota) = val'(\pi_1(\iota)) \circ val(\pi_2(\iota))$. Then we say that $\mathcal{I}$ **satisfies** a set S of entity class schemes (and in particular, an IDEF1 diagram) iff

1. For all $\sigma \in S$,

   1.1. For all $\iota \in IAN(\sigma), dom(val'(\iota)) = val(ECN(\sigma))$;

   1.2. For all $\omega \in OAN(\sigma), dom(val(\omega)) = val(ECN(\sigma))$;

   1.3. For all $\kappa = (\alpha_1, \ldots, \alpha_n) \in KCN(\sigma)$, and f or all $x, y \in val(ECN(\sigma))$, $\langle val'(\alpha_1)(x), \ldots, val'(\alpha_n)(x) \rangle \neq \langle val'(\alpha_1)(y), \ldots, val'(\alpha_n)(y) \rangle$, and there is no $i \leq n$ such that for all $x, y \in val(ECN(\sigma)), \langle val'(\alpha_1)(x), \ldots, val'(\alpha_{i-1})(x), val'(\alpha_{i+1})(x), \ldots, val'(\alpha_n)(x) \rangle \neq \langle val'(\alpha_1)(y), \ldots, val'(\alpha_{i-1})(y), val'(\alpha_{i+1})(y), \ldots, val'(\alpha_n)(y) \rangle$;.

2. For all $\sigma, \sigma' \in S$, if σ is linked to σ' via λ, then $rng(val(\lambda)) \subseteq val(ECN(\sigma'))$.

8.1.2 IDEF1/ES Method Description

One of the critical tools needed to implement the KAMSS concepts presented in Section 3 is a knowledge acquisition method to be used by both the initial builders of the KAMSS, by the installer of KAMSS at a particular facility, and by personnel responsible for the maintenance of the knowledge bases and system description bases. In this section we describe such a method derived from the existing IDEF1 information modeling tool [Ramey 1981]. This method, referred to as IDEF1/ES, was designed for human use during the analysis of manufacturing system description texts and construction of the requisite knowledge bases. It was designed to be consistent with the situation semantics presented in Section 5. The Air Force IDEF1 methodology was chosen as the base for this method development because of the widespread familiarity with this methodology which exists in the manufacturing community, and because the theory on which it was based was the easiest to extend to include the necessary concepts.

The IDEF1/ES reflects our experience base with respect to knowledge acquisition needs for representation support [Friel and Mayer 1985]. The elements of this methodology were motivated by the ontology developed in Section 5 of this dissertation. This methodology can adequately capture the information necessary to drive the SDCE components of the KAMSS. Finally, this methodology was designed to be able to represent the domain knowledge in the prototype systems described in [Mayer 1986; Friel 1987; Krishnamurthi, et al 1986a, b, c; Sterle, et al 1986; Mayer, et al 1987].

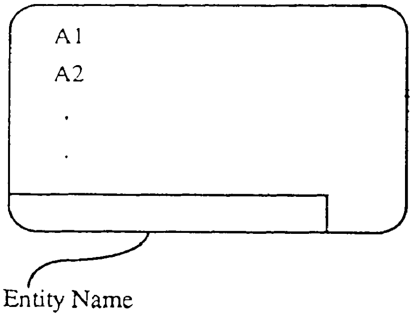
The basic intuition behind IDEF1/ES is that our interaction with the manufacturing system occurs at several levels. At one level, we acquire information about the physical word via our sensory mechanisms. At another we acquire and structure information at a concept level through abstraction and generalization and via communication with other humans. At yet another level, we acquire information via symbol sets accumulated by the information system in the environment. The original IDEF1 was designed to model the last of these views. Our extensions in IDEF1/ES provide the capability to represent the other views and the relationship between these views. As well, the constraint specification extensions for both views provide the needed mechanisms for capturing the "behavior enabling" knowledge which determines the dynamics of the system over time.

8.1.2.1 Entity Representation Extensions

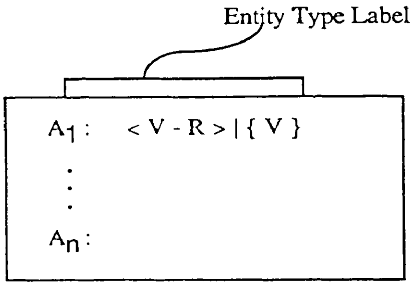
As previously mentioned, the primary extensions to the concept of an entity are the introduction of the notion of entity types, named instances, named members, and described types/instances and described classes/members. The graphical syntax introduced for these extensions is displayed in Figure 8.2 and 8.3.

The semantics of the entity type concept is a rather complex extension to the original semantics of IDEF1 presented above. The original IDEF1 only recognized the existence of "class" concepts. A "class" in the original IDEF1 is a "set" denoted by the entity class label and defined intensionally as a set of attribute classes, having an extension which was a set of sets of attribute–value pairs. This was an important departure from traditional data modeling constructs and also a departure from existing information modeling constructs at the time. The existing methods used the members of a set to represent the physical objects themselves. Thus, an employee relation represented a "set" of employees. Actually

NAMED ENTITY INSTANCES SYMBOL



ENTITY TYPE SYMBOL

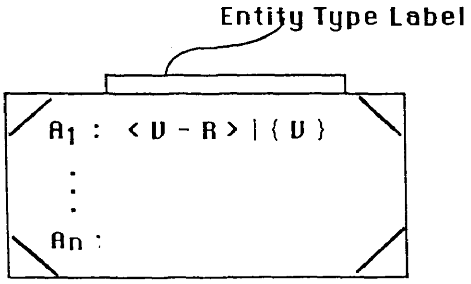


- May Inherit Attributes or Properties from Other Entity Types

- May Be Source of Property Inheritance for Entity Classes, Instances, & Other Types

FIGURE 8.2: IDEF1 SYNTAX EXTENSIONS FOR INSTANCES AND TYPES.

ENTITY TYPE SYMBOL



- May Inherit Attributes or Properties from Other Entity Types
- May Be Source of Property Inheritance for Entity Classes, Instances, & Other Types

FIGURE 8.3: IDEF1 SYNTAX EXTENSIONS FOR DESCRIBED ENTITIES.

even though this is the way it is described, in actuality an intensional sort of thing was meant. One whose extension at any point in time was to be a set. The important point which must be understood in order to justify the entity type concept is that the intentional definition of the IDEF1 entity class is meant to be fixed. In other words the attribute classes which contribute to the definition of an entity class are assumed not to change over time. Thus an entity class can be viewed as the sort of thing whose intensional definition is a set whose extension is fixed (it is the set of attribute use classes which are displayed by that entity class, nothing more, nothing less). The extension of an entity class at any point in time was intended to be a set each of whose members were a set of pairs of displayed attributes and a value. The notion of an entity type relaxes this restriction.

The basic idea of introducing the concept and modeling construct for a "type" is to allow the representation of concepts in a system description which are inherently ill defined. The rationale for introducing such a construct is that many of the "rules of operation" in an environment are based on such ill defined concepts (if this were not the case there would be no real need for a judicial system!). It is important to note that a type, cannot be modeled consistently as a "set". Not only does it lack a consistent extension, but also it lacks a consistent intension. Thus while one can name a type one cannot consistently describe or characterize a type. This would appear to make it a fairly slippery and possibly useless concept. However one can make assertions about a type, and herein lies its value. It also turns out that if one recognizes which assertions can be made about types versus those which can only be made about classes, many of the traditional paradoxes associated with semantic nets, and many of the limitations associated with information models disappear. The rules governing link participation introduced in IDEF1/ES are designed to restrict the user to avoid such paradoxes, as will be described in a later subsection on relation restrictions. One of the other important roles which the type concept plays is in the specification of the source of descriptive attribute classes within an entity class. In the original IDEF1, the only attribute classes which could be inherited across a link were members of the key attribute classes. With the introduction of types in IDEF1/ES came the ability to define a number of specialized link types. One of those link types is the "characteristic type" link. Via a link of this type between an entity type and an entity class, one can specify the source of descriptive attribute classes in the entity class. The semantics of a type instance similarly depart radically from the IDEF1 notion of entity (entity class member).

The entity member extension is an attempt to elevate entity members to "first class" model elements. In the computer science jargon relative to language theories, "first class" elements are those language elements which can be directly referred to as well as used. Thus for example, in Lisp, "functions" are first class elements as in Lambda calculus. In traditional IDEF1, the concept of entities as information images of real world objects was obviously recognized; however, they could not be directly referred to. Only the classes of information about them could be addressed. This restricted the ability of the IDEF1 methodology to make assertions about the properties of a specific individual image. The limitation of such a restriction becomes particularly evident when attempting to express specific rules in a system description such as *If the "production order" for the part is complete then a "work order" is issued.* The type instances provide the extension to the IDEF1 modeling methodology that allows one to directly reference physical or conceptual objects. It should be noted that an entity class member is not the same as a type instance! What is displayed in the entity class member is the same information as one would find in the set of attribute - value pairs (associated with, assigned to, or observed upon) the type instance by the organizations information system. The difference is in what the displayed information stands for. In the case of the type instance, the information stands for the particular attributes of the referenced object. In the entity class member, the information stands for itself (the name of the thing is the thing named).

The need for the capability to represent concepts referred to by descriptive reference is again in support of the need to represent the kinds of constructs found in the sample system descriptions. It is quite common for a person to refer to concepts for which there is no special name (or label) but rather only a description. For example:

1. "Semi-finished parts,"

2. "Approved production orders,"

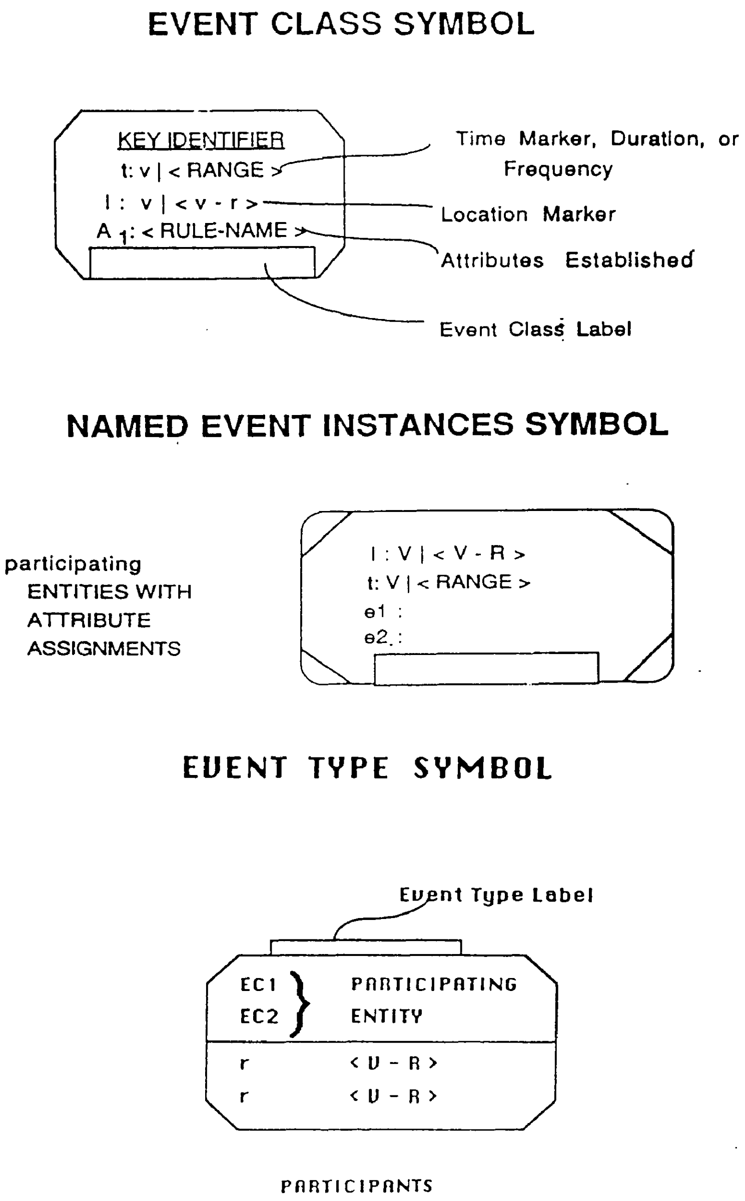
3. "Non-conforming parts awaiting rapid rework".

The issue associated with such a reference is, of course, to what the reference refers (i.e., the describing conditions or the described object). To remain consistent with the IDEF1 entity class scheme, the choice for the referent of a definite description of an entity class or entity class member is the describing conditions. Similarly for the referent of a definite description of a type. However, the choice for the referent of a definite description of a type instance is the described object. The intended use of definite descriptions is to serve as convenient mechanisms for the description of antecedents of rule classes. They are also useful in the representation of state change conditions and as links between entity classes and event classes.

8.1.2.2 Event Representations

One of the key shortfalls of IDEF1 semantics needed for knowledge acquisition is the capability to represent time and location staked situations and states of affairs. This shortfall also restricts the ability of the IDEF1 method in the representation of conditional constraints which must reference these structures. The introduction of classes, types, and instance structures of events represents our attempt to fill these voids. Figures 8.4 and 8.5 display the syntax for event structures. The following paragraphs will describe the semantics of those structures.

1. **Event Classes:** The notion of an event class is a natural extension to the notion of an entity class. It provides a means of representing change. Just as the entity class represents information which is managed by the organization and not either all of the information in an organization or all of the objects in the environment of the organization, event classes represent the set of information which is managed or maintained about change. Thus, the notion of event classes is modeled as an intensionally defined set. The definition of that set is comprised of a set of attribute classes. Like the entity class, an event class must have an attribute class which assumes a value which is unique for each instance in the event class. Unlike the entity class, the composition of the key attribute class is structured. That is, there is the need for not only a unique identifier but also a stake reference of some variety (i.e., time reference, location reference, etc.) The event class symbol used in IDEF1/ES is shown in Figure 8.4. An event class is defined



EVENT CLASS SYMBOL

NAMED EVENT INSTANCES SYMBOL

EVENT TYPE SYMBOL

May Inherit Participants, Properties, and/or Their Restrictions from Other Event Types

May be Source of Property or Participants for Event Classes

FIGURE 8.4: NAMED EVENT CLASSES, INSTANCES, AND TYPES.

DEFINITE DESCRIPTIONS OF EVENT INSTANCES SYMBOL

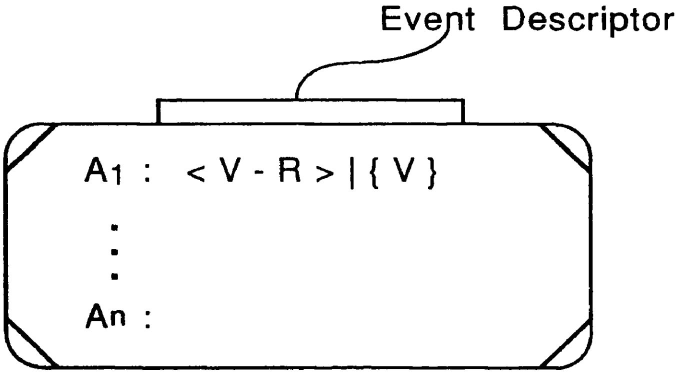


FIGURE 8.5: DESCRIBED EVENT CLASSES, INSTANCES, AND TYPES.

to involve a fixed set of entity classes (i.e., what is indexed over what is the stakes (e.g., time) and the attribute values of a fixed set of entity classes).

2. **Event Types:** The notion of an event type is introduced to allow the representation of change concepts which extend over many situations. The event type can be thought of as an event class where the participating entity classes are undefined. For example, the event type allows the modeler to represent a generic "creation event" without saying anything specific about the information which is "managed by the environment" about that sort of change. Thus, event types provide for "change information" specification while the entity types provided for "static information" specification.

3. **Event Instances:** Finally, the event instance concept is introduced to allow for the representation of a specific real world event. Note that event instances must refer to entity instances and hence to objects in the real world. Thus, an event instance is not the same as an instance of an event class. The latter refers to a collection of information not the actual event.

4. **Described Events:** Just as in the case of entities, there are many situations where events are referred to by descriptive reference. To allow for the representation of such phenomena, a syntax is provided for described event classes, types and instances. The IDEF1/ES syntax for these representations is displayed in Figure 8.5.

8.1.2.3 Rule Collection

Rule collection allows the represention of conditionals and conditionally specified constraints which fall outside of the standard constraint set specified by the IDEF1/ES link types as outlined further. In essence, one can consider the built in link types as predefined macros of rule collections. These prepackaged rule collections were chosen to allow the easy representation of the most commonly used constraints. The restrictions associated with their definitions also help insure the proper semantics of a specification. However, it should be noted that all of these predefined link types could be defined as rule collections. The symbol used to represent rule collections is displayed in Figure 8.6. The "rule name" is used both as a reference and as an index into a backup form which actually contains the rules. The "rule name" is optional in that the actual rule can be inserted into

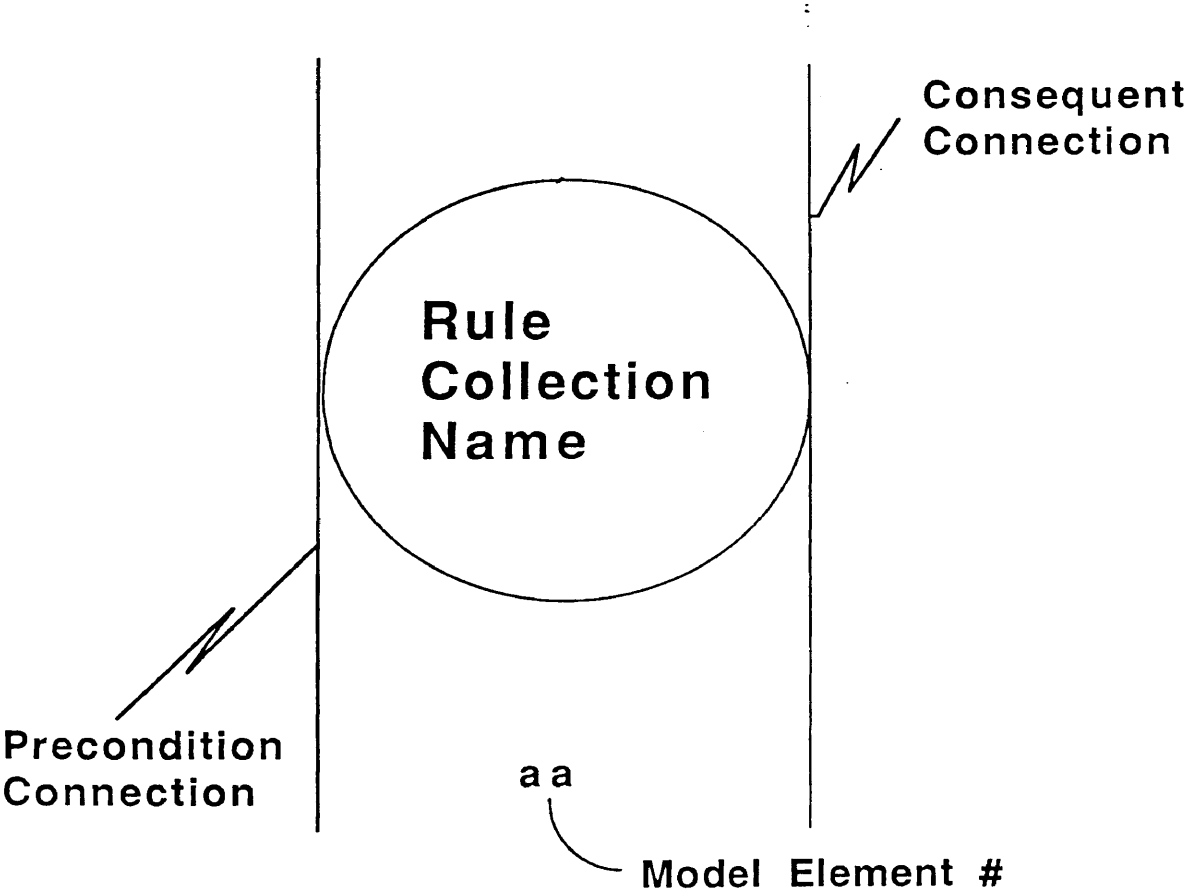


FIGURE 8.6: SYNTAX FOR RULE COLLECTIONS.

the circle. This is to allow for use of "un-named" rules (similar to Lisp Lambda forms).

The design of a rule language for IDEF1/ES required the definition of a textual language for the constructs in the IDEF1 as well as the IDEF1/ES extensions. That is, in order to speak about the existence of an entity in an antecedent to a rule or the attribute value of an attribute within a particular entity class in a textual form, we need a representation structure for those concepts which is not graphical. Actually through the use of the "pointer" link types as described in the next section, we can indicate the participation of such objects graphically. However, for complex models or complex rules, such graphical representations become very obscure. A nongraphical syntax is also helpful in the automation of the analysis of the information in these models.

Another aspect of rule specification is applied rule application control. That is, when is a rule applied, how is it to be applied, and over what domain should it apply when in fact it is applied. In the IDEF1/ES, these control specification problems can be considered as operating at two levels. The first level is within a rule set itself. The second level is on the rule set level. When there is only one rule in the rule set, the control of the rule set application defaults to the rule set application control. The "prologue" section of the rule set specification is used to specify the control stategy to be applied. One can think of this control logic as instructions to the individual IDEF1/ES model objects telling them how and when to execute the encapsuled rules. Thus, the rule sets can be thought of as becoming "attached" to a conceptual object which is composed of the types, classes, and instances which participate in the condition sets of its member rules. Such rule sets fire when they are told to fire. Thus, they can be considered to be procedural in nature. The other class of rule activations which we want to be able to represent are those rules which are conceptually non-procedural in nature. Such rules can be thought of as being written as instructions to an overall concept/information manager. The representation of this rule type requires the specification of the rule selection and application logic of the concept manager, as well as the identification of which rule sets fall into its domain.

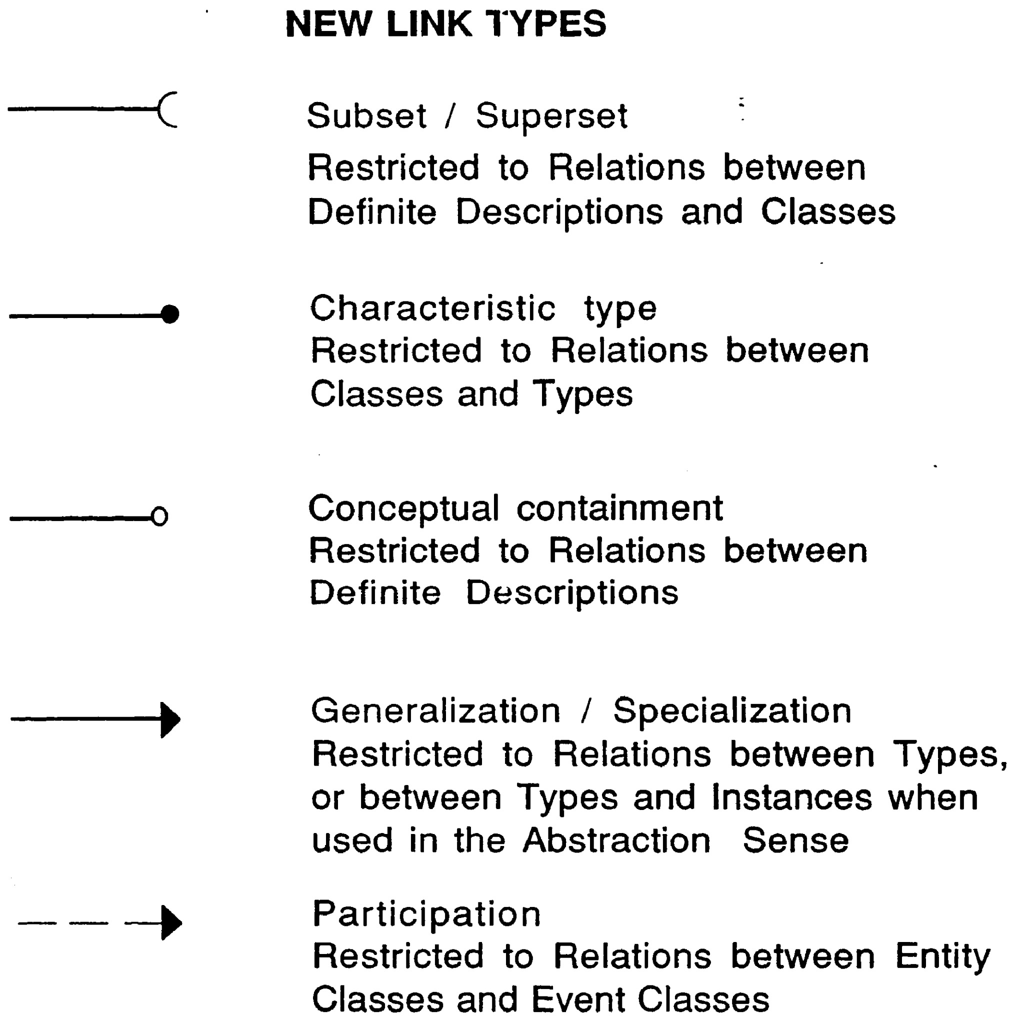
There are two syntaxes for rule specification. One is English-like, allowing both indicative and subjective conditional specifications. The other is modeled after the production rule languages [Inference 1985; CLIPS 1986; Carnegie Group 1986; Stefik 1986].

8.1.2.4 Predefined Link Types

As mentioned in the preceeding section, there are a number of predefined rules which are available to the modeler as link types. These links carry with them the necessary constraints for use that help to insure proper application by the modeler. Such application constraints, if followed, prevent most of the logic paradoxes which have plagued semantic net modeling methods over the past 15 years [Brachman 1983]. Figures 8.7 and 8.8 display the current set of link types provided in IDEF1/ES. One of the link types provided (the *pointer* in Figure 8.8) required the redefinition of the non-specific one-to-one link of IDEF1. Outside of this redefinition, all of the previous link types of IDEF1 are still available. The following items provide a description of the semantics of the new link types.

1. **Subset / Superset Links:** Because of the recognition in IDEF1/ES of the distinction between classes types and instances, there are a number of natural restrictions which can be placed on the use of the subset / superset links. For example, one cannot say that a class is a subset of a type since a type is not modeled as a set in any fashion. Also since the semantics of an instance is the described object and the semantics of an instance of a class is a set of symbols the subset / superset relation does not apply to the relations between instances and classes. In fact, the only places where the subset / superset relation can be used is in the relationship between classes (i.e., entity classes, and event classes of the named or described variety).

2. **Characteristic Type Links:** The characteristic type link is provided to show the type of binding which can exist between a class and a type. It means to represent that the information displayed in an instance of a class is information carried by the type concept. Unlike the link between entity classes which denote existential relationships, the characteristic link does not carry any such connotation. Note that because of the definition of the semantics of an IDEF1 class, there cannot be a characteristic type link between a class and another class. It is possible, however, for an instance to serve as the characteristic type of a class.



NEW LINK TYPES

Subset / Superset
Restricted to Relations between
Definite Descriptions and Classes

Characteristic type
Restricted to Relations between
Classes and Types

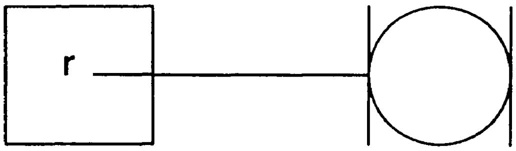
Conceptual containment
Restricted to Relations between
Definite Descriptions

Generalization / Specialization
Restricted to Relations between Types,
or between Types and Instances when
used in the Abstraction Sense

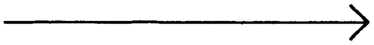
Participation
Restricted to Relations between Entity
Classes and Event Classes

FIGURE 8.7: NEW LINK TYPES.

Links in the Box



Association
Used to Denote Rule
Collections as Value
Restrictions on
Property Values



Instance Restricted
to Relations between
Named Objects and Classes

FIGURE 8.8: NEW LINK TYPES CONTINUED.

3. **Conceptual Containment Links:** This is a type of link which is used to express that one description includes another. This type of link is restricted to use with descriptive classes, types, and instances only. When used, it refers to the descriptive component only of these concepts (i.e., the label inside the header box.)

4. **Generalization / Specialization Links:** These types of links are used to express relations between types and between types and instances. When used between types, the generalization links capture the semantics of "is a" and "is a kind of" relations between types. Such links can be used to construct type hierarchies or general type networks.

5. **Participation Links:** As mentioned in the previous section on event representation, an event class must involve at least one entity class. The participation links are available as a mechanism for illustrating the relationships between event classes and entity classes.

6. **Links in a Box:** Because of the addition of the event constructs and the rule sets each with participating entity classes, there is a need to provide links which allow one to point into classes, types, and instances to indicate participating or affected attributes. These pointer links have no inherent semantics, rather they just serve as convenient notational devices.

7. **Instance Links:** Instance links provide a means of relating instances to classes and types. An instance link between an instance and a type is a form of a specialization link. The semantics being that the instance is "AKO" (e.g., a kind of) related to the type. The semantics of the instance link to the class indicates that the information modeled in the class structure serves as a prototype for the instance within the formal information system. It does not indicate that the instance is an instance of the class as discussed above.

8.1.2.5 Links Between Links

One of the differences between IDEF1 and IDEF1/ES as discussed in a previous section was the recognition of "entity instances" as first class model elements. In addition to this, we have also allowed the links described in the previous section and by default, the rule sets as well, to stand in relations and hence be referable to. This allows the modeler to express an entire new class of constraints relative to the participation of entities in the existing links. The set of constraints provided (with the associated symbols) are displayed in Figures 8.9 through 8.13. Most of these constraints types have been identified as necessary by the original developers of IDEF1 [Ramey 1983] and ENALIM [Nijssen 1982].

1. **Inclusively Independent Constraints:** The inclusively independent constraint allows one to represent the situation where a single relation maps a dependent entity class into multiple independent entity classes simultaneously (see Figure 8.9).

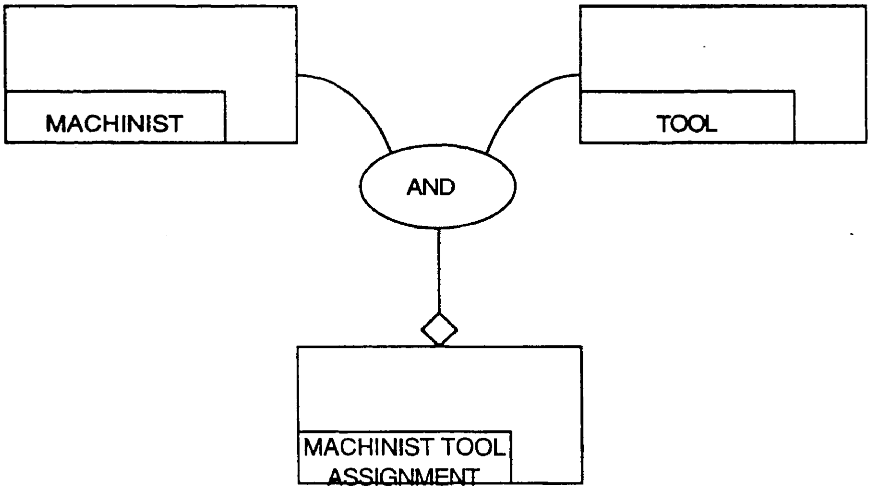
2. **Exclusively Dependent Constraints:** The exclusively dependent constraint allows one to represent the situation in which two dependent entity classes share a common range but not simultaneously. Thus, in Figure 8.10 a bond issue is either a corporate bond issue or a municipal bond issue but never both.

3. **Exclusion Constraints:** An exclusion constraint between two relations that share a common range and a common domain implies that an element in the domain cannot be mapped by both relations to the same element in the range. Thus, in Figure 8.11, an employee cannot be his own backup on a job assignment.

4. **Uniqueness Constraints:** A uniqueness constraint between two relations that share a common domain implies that two different elements in the domain which map to the same element in the range of one of the relations cannot map to the same element in the range of the other relation. For example, in Figure 8.12, the uniqueness constraint implies that an employee can only work on one job in a project or conversely, that a project will have different employees on each job.

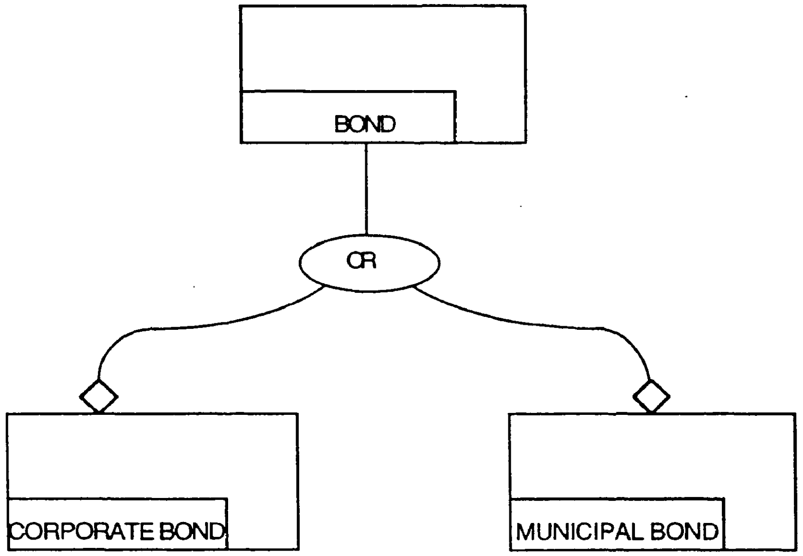
5. **Subsetting Constraints:** A subsetting constraint between two relation classes which share a common range implies that if one relation instance is realized, then the other must also be realized. For example in Figure 8.13, an employee who has a child must have a dependent. Note that the syntax indicates the direction of the existential dependency (i.e., an employee may have a dependent who is not a child).

INCLUSIVELY INDEPENDENT SYMBOL



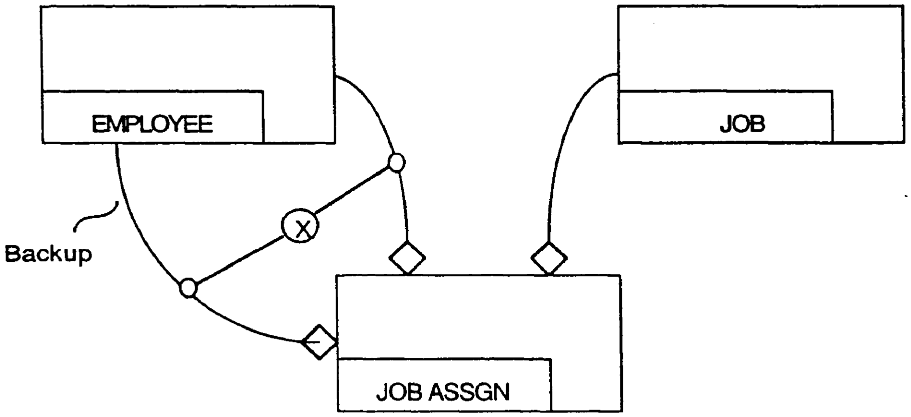
Reading:
A machinist tool assignment requires
both a machinist and a tool.

FIGURE 8.9: INCLUSIVELY INDEPENDENT CONSTRAINT ON RELATIONS.



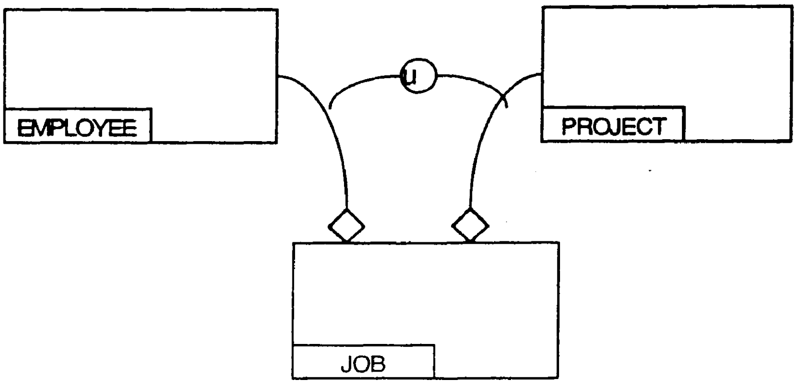
Reading: A bond issue is either a corporate bond or a municipal bond.

FIGURE 8.10: EXCLUSIVELY DEPENDENT CONSTRAINT ON RELATIONS.



Reading: An employee cannot be his own backup on a job assignment.

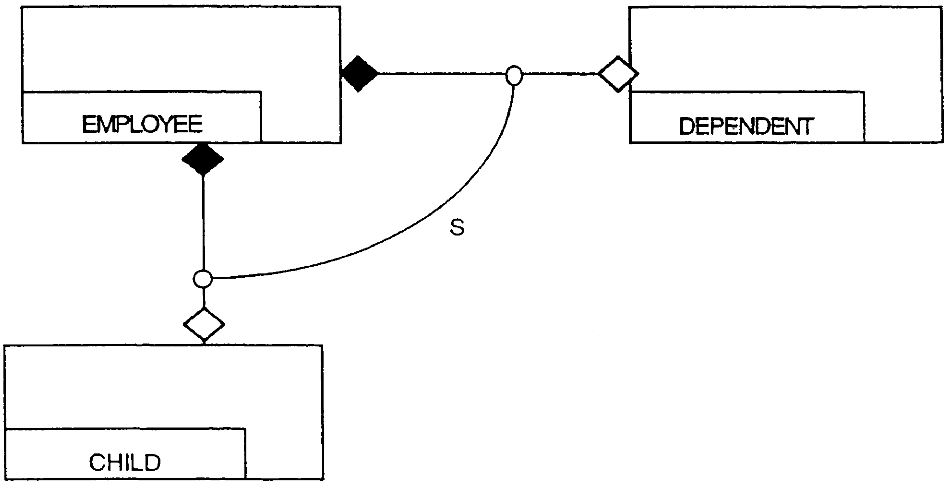
FIGURE 8.11: EXCLUSION CONSTRAINT ON RELATIONS.



Reading:

An Employee can only work on one Job in a Project.

FIGURE 8.12: UNIQUENESS CONSTRAINT ON RELATIONS.



Reading:

An Employee who has a child must have a dependent.

FIGURE 8.13: SUBSETTING CONSTRAINT ON RELATIONS.

8.1.2.6 Examples of IDEF1/ES Application

Figure 8.14 provides an example of the use of the above described concepts of IDEF1/ES.

8.1.2.7 IDEF1/ES Conclusions

We have developed a concept modeling tool which can be used to represent the knowledge and information which is necessary to support:

1. The analysis required to design natural language processing interfaces to the KAMSS,
2. The knowledge engineering to design the knowledge base components of the KAMSS subsystems,
3. The information engineering necessary to design the database components of the KAMSS.

By building from an existing information modeling methodology, we believe that we will minimize the necessary relearning required to use the proposed method. This is particularly important because the construction of knowledge based systems in manufacturing domains is done simultaneously with major initiatives in information integration. The ability to build from models already constructed in the gathering of information for a KAMSS implementation is critical in speeding its introduction into the manufacturing environment. In fact, the extensions encompassed in the IDEF1/ES address all of the shortfalls which have been identified with current information modeling methods [Mayer et al. 1988]. As such, there is the possibility that it could replace the existing modeling techniques providing a uniform method for both information and knowledge engineering.

8.1.3 IDEF3 Method Description

One of the primary mechanisms used for description of the world is relating a story in terms of an ordered sequence of events or activities. The original IDEFs were developed for the purpose of enhancing communication among people who needed to decide how their existing systems were to be integrated. IDEF0 was designed to allow a graceful expansion of the description of a system's functions

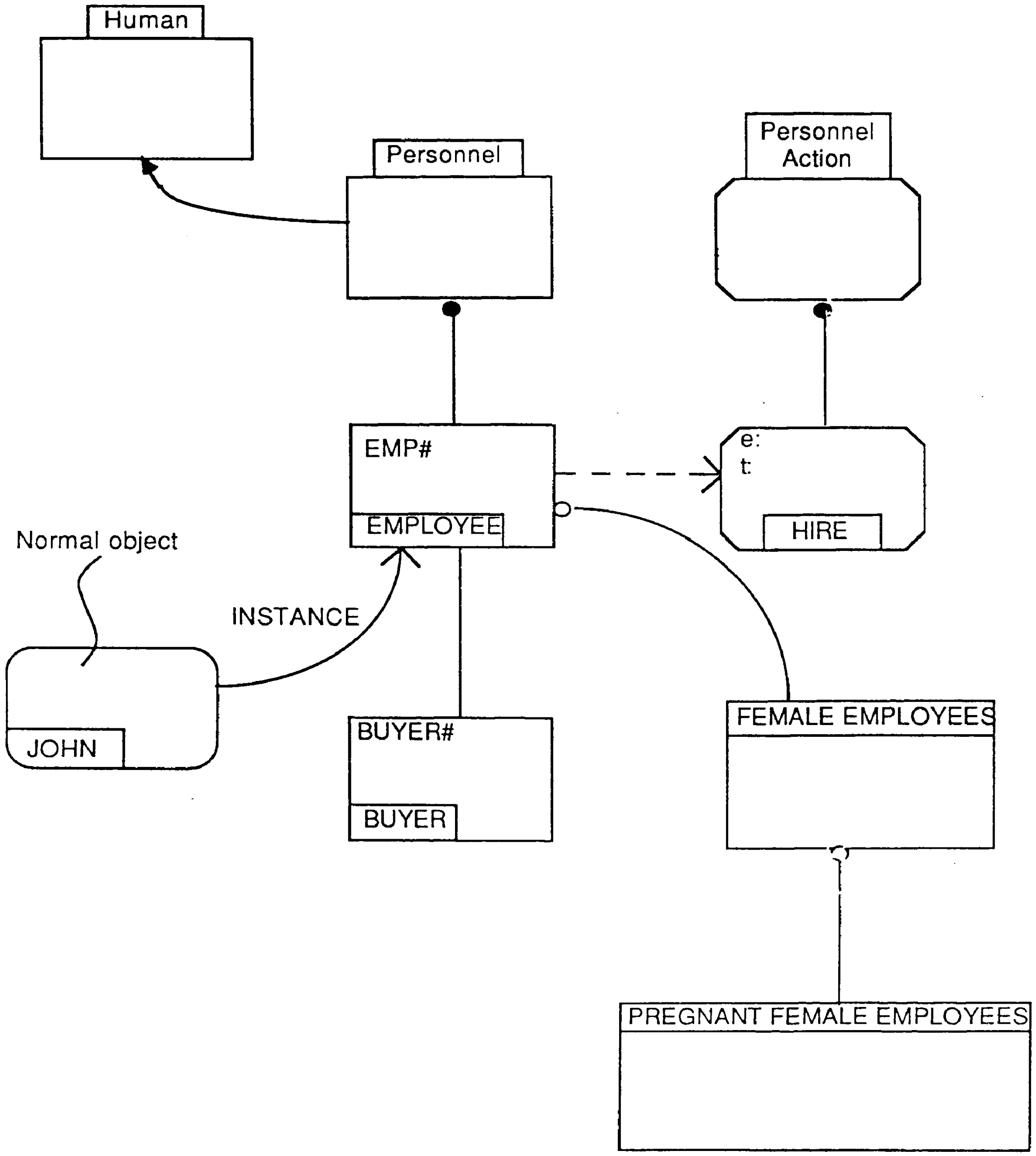


FIGURE 8.14: EXAMPLE OF TYPES, CLASSES, DESCRIPTIONS, AND INSTANCES.

through the process of function decomposition and categorization of the relations between functions (i.e., in terms of the Input, Output, Control, and Mechanism classification). IDEF1 was designed to allow the description of the information that an organization deems important to manage in order to accomplish its objectives. The third IDEF (IDEF2) was originally envisioned as a user interface modeling method. However, since the ICAM Program needed a simulation modeling tool, the resulting IDEF2 was a method providing a framework for specification of mathematical simulations. It was the intent of the methodology program within ICAM to rectify this situation, but limited funding did not allow this to happen. As a result, the lack of a method which would support the structuring of descriptions of the user view of a system has been a major shortcoming of the IDEF system. The basic problem from a methodology point of view is the need to distingush between a **description** of what a system (existing or proposed) is supposed to do and a **representative simulation model** that will predict what a system will do. The latter was the focus of IDEF2, the former will be the focus of IDEF3.

In many situations, users of the IDEF0 method have resorted to special extensions to the IDEF0 syntax and technique in order to modify this method to satisfy the need to represent the logical progression of activities which describe the "behavioral" characteristics of the modeled system.

In relation to the KAMSS, the IDEF3 provides a framework for entering these descriptions of process flows and object states in a manner which minimizes the error in making such descriptions and which assures the viability of the qualitative reasoning analysis described in Section 7.

8.1.3.1 Requirements for IDEF3

The previous discussion of the motivation behind the definition of a new method for process flow and object state description points to two specific needs. One need is for a method to support the development of a mechanistic description of "how" a system works. That is, how a set of components is organized to solve a particular type of problem. Another is the need to describe "how" an existing or proposed organization of activities accomplishes a desired sequence of states of affairs. Additionally, a third need is for a method to support the description of what the "interaction" will be between agents and the objects acted upon in the system. Agents in this sense may be humans or any other system component.

IDEF3 must provide the concepts, syntax, and procedures for building *requirements models* which are descriptions of a system adequately detailed to determine if a system described will actually work.

8.1.3.2 IDEF3 Design Concepts and Philosophy

There are a number of problems associated with any attempt to describe change. Some of these problems are definitional in nature such as defining terms as "process," "event," and "activity" as well as characterizing the difference between these concepts. Presuming a consistent definition, the next stumbling block comes from the difference between the orderings and relations which can be established between "types" of such concepts and those which apply to "instances" of such concepts. For example, simple relations like "before" which can unambiguously be applied to event instances (e.g., I ate supper before I typed this section) require considerable elaboration if applied at the "type" level. If one activity type precede (follows, overlaps, etc.) another activity type, it may not necessarily imply that all possible instances of one activity type precedes all possible instances of the other. More often, it means that the instances of the two activities are pairwise related such that each instance of one activity precedes a corresponding instance of another activity. Such ambiguity can be resolved with additional specification mechanisms although there is a trade-off of understandability versus accuracy.

Unfortunately as the level of specificity goes up, there is a corresponding decrease in the comprehensibility of the models. The basic situation is an untenable one. Descriptions of large systems can either be accurate or comprehensible. In Figure 8.15, we see one classical way of dealing with the complexity of descriptions involving process specifications and object states. In Figure 8.15a, we see a representation of the totality of the information which we are attempting to relate as a three space of objects, their conditions, and time. In Figure 8.15b, the hexagons represent external events that start a process flow or create an object in a condition. The circles represent objects in various conditions. The squares represent activities which change an object's condition. The horizontal flows illustrate a path where the object is changing based on the performance of the

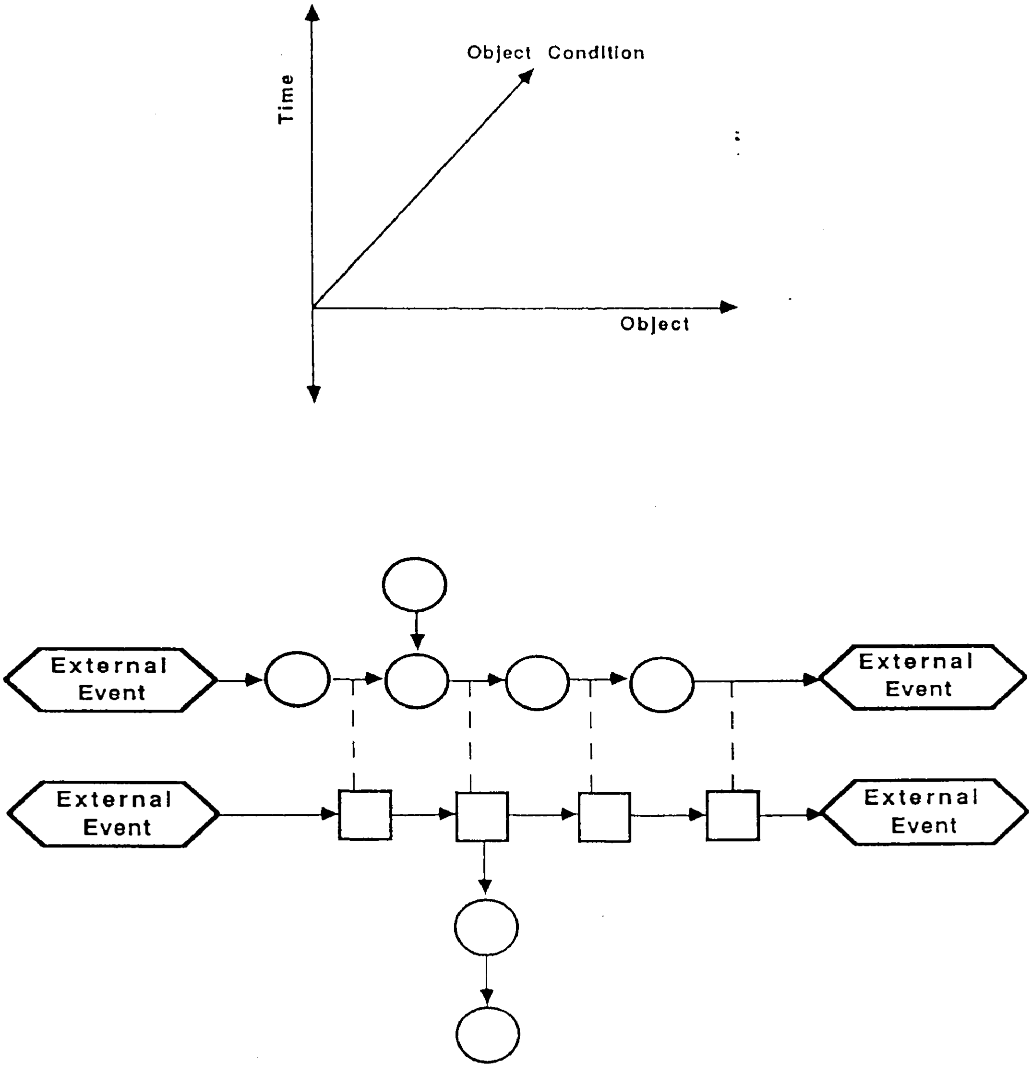


FIGURE 8.15: INTERACTION OF OBJECT, TIME, AND OBJECT CONDITION.

actions (represented by square boxes with dotted lines showing the transition the activity is responsible for). One example of this is the change of a purchase request into a purchase order. The vertical line of circles represents changes in the condition of a particular object (for example the change of a proposed purchase request into an accepted purchase request).

Complexity also can be controlled through introduction of views, levels, and properties. We also note that in practice, adequacy is given precedence over accuracy in the description of process flow. Therefore, we set as a design goal for the IDEF3 that the modeler should be able to gracefully introduce complexity and expose that complexity as necessary in the syntax of the modeling system. To this end we will start with as complete a characterization of the basic concepts of the IDEF3 method but introduce a simple syntax for the initial model construction. To a certain extent this approach to modeling parallels the IDEF1 method with its five phases. However, the approach we are attempting designs the abstraction into the modeling concepts themselves rather than into the discipline.

8.1.3.3 Basic Concepts of IDEF3

The basic concepts of IDEF3 include: space, time, situation, event, action, process, state, and conditional. Around these basic concepts, we are building a number of primitive relations which are useful for describing most of the characteristics necessary to capture in the description how the activities, agents and objects integrate into an operational scenario. The basic relations fall into the following categories:

1. Temporal relations,

2. Sequence relations,

3. Causality relations,

4. Containment relations,

5. Typing relations.

The syntax for expressing process flow and object state is based on these concepts. As expected, we will normally represent instances of the concept types with a node symbol and instances of the relation types with links between the node symbols. Where possible we will use the position of the nodes on a page and the left / right ordering of the nodes to indicate sequence relations.

8.1.3.4 IDEF3 Syntax

There are actually two different model types in IDEF3, these are the process model and the object state model.

The following subsections describe a proposed syntax for the process flow and object state models. This syntax supports the creation of the individual models as well as the indication of the integration links between these submodels and the IDEF0 and IDEF1 models. Because of the complexity of the collection of these models as well as the individual models themselves, it is obvious that automated tools will be required to effectively handle comprehensive models of this sort. However since such tools will not be available at all times, some practical rules of thumb and guidelines for constructing these models manually and tracking the information included within these models will be provided.

As with the IDEF0 and IDEF1 methods, a process flow model must be accompanied by a statement of viewpoint, purpose, and context. In addition an IDEF3 model will have a "scenario" label. The descriptions of these four textual components of an IDEF3 model are given in Figure 8.16. The relationship between an IDEF3 model set and IDEF0 or IDEF1 is dependent on the set-up of the viewpoint, purpose, and context for the three models. For a meaningful integratation with existing IDEF0 and IDEF1 models, careful coordination of the viewpoint, purpose, and context must be maintained across all models.

8.1.4 Illustrating the Process Model

The process model captures a network of relations between actions in a specified scenario. The basic symbol set of the IDEF3 process model is displayed in Figure 8.17. The structural component is based on the conventions of the "Graphical BNF" diagramming technique (as recommended by Tim Ramey). Though not displayed in Figure 8.17, each model element in the process model carries the following attributes:

1. Name (unique across the process flow),

Viewpoint Statement:

Characterizes the perspective of the information presented in the model.

Purpose Statement :

Describes the intended use of the model.

Context Statement:

Establishes the boundary of the model. In IDEF3 the context statement functions primarily as a means of classification of events as being internal or external to the system.

Scenario:

The label serving a the name for the process being described in the organization.

FIGURE 8.16: VIEWPOINT, PURPOSE, CONTEXT, AND SCENARIO DESCRIPTIONS.

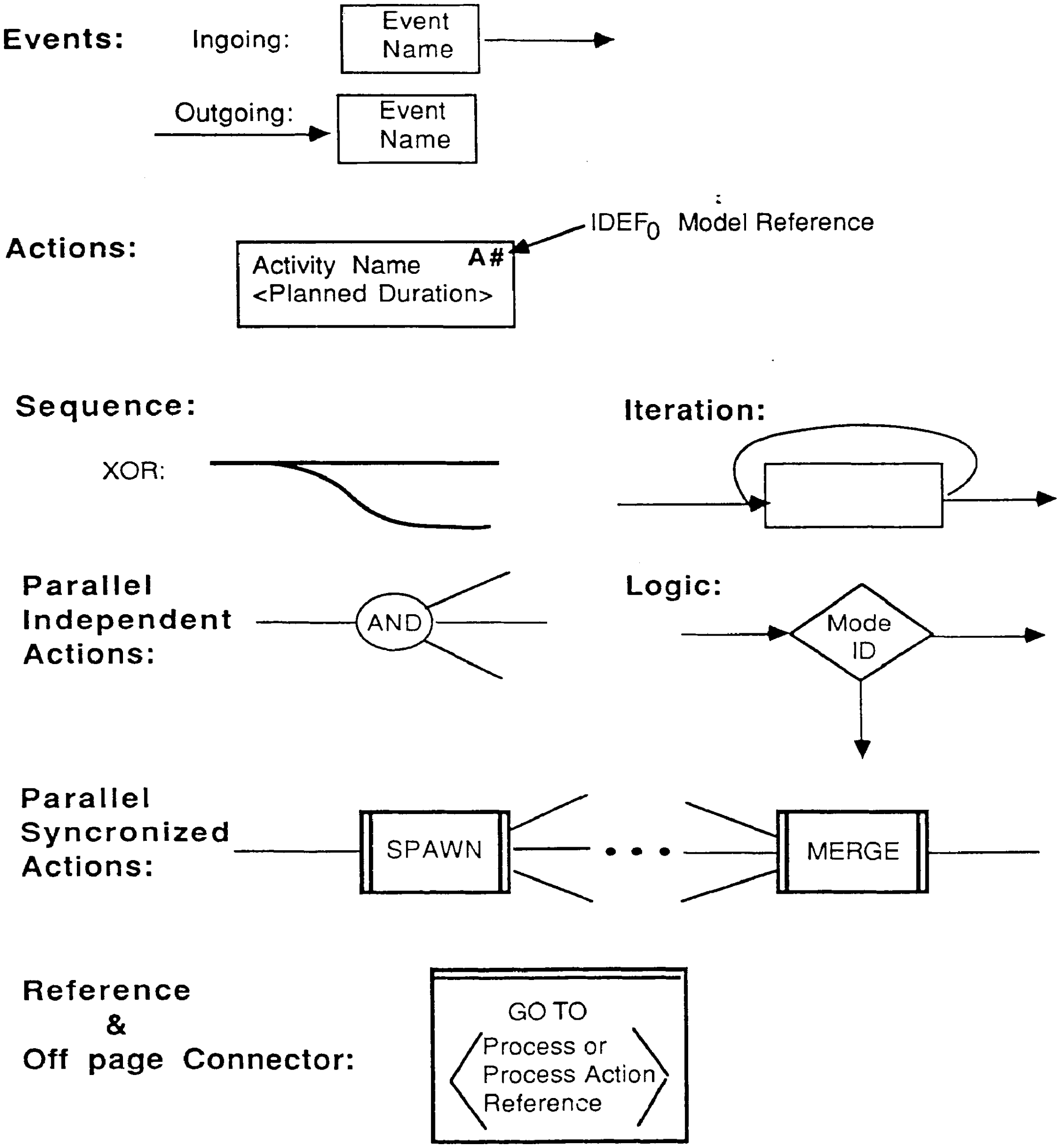


FIGURE 8.17: PROCESS NETWORK SYMBOL SUMMARY.

2. Label (displayed with the symbol),

3. Description (a glossary entry).

8.1.5 Object State Transition Description

The basic form of the object state transition diagram at this point in time is that of an augmented transition network. Figure 8.18 displays the basic elements of the object state model. The nodes (solid boxes) in the network represent object states. The dashed boxes attached to the nodes contain lists of attributes whose specified values must be realized before a transition can be attempted. Note that the attribute-value pairs may contain entries that are not in the object state. The labels on the arcs are names of other object transition networks. The meaning of a labeled arc is that the corresponding object state transition must be completed before the transition can be completed. Though not displayed in Figure 8.18, each model element in the process model carries the following attributes:

1. Name (unique across the process flow),

2. Label (displayed with the symbol),

3. Description (a glossary entry).

8.2 SDCE: A System Description Capture Environment

The System Description Capture Environment (SDCE) is a development effort targeted at establishment of a knowledge based system for the:

1. Collection,

2. Organization, and

3. Analysis

of manufacturing knowledge in a plant. The SDCE prototype provides a proof of concept for the SDC part of the KAMSS. The SDCE provides the expertise of an experienced manufacturing systems analyst to assist the decision maker in organization of data about his manufacturing facility. This base of information supports the automatic generation of simulation models thus reducing the time

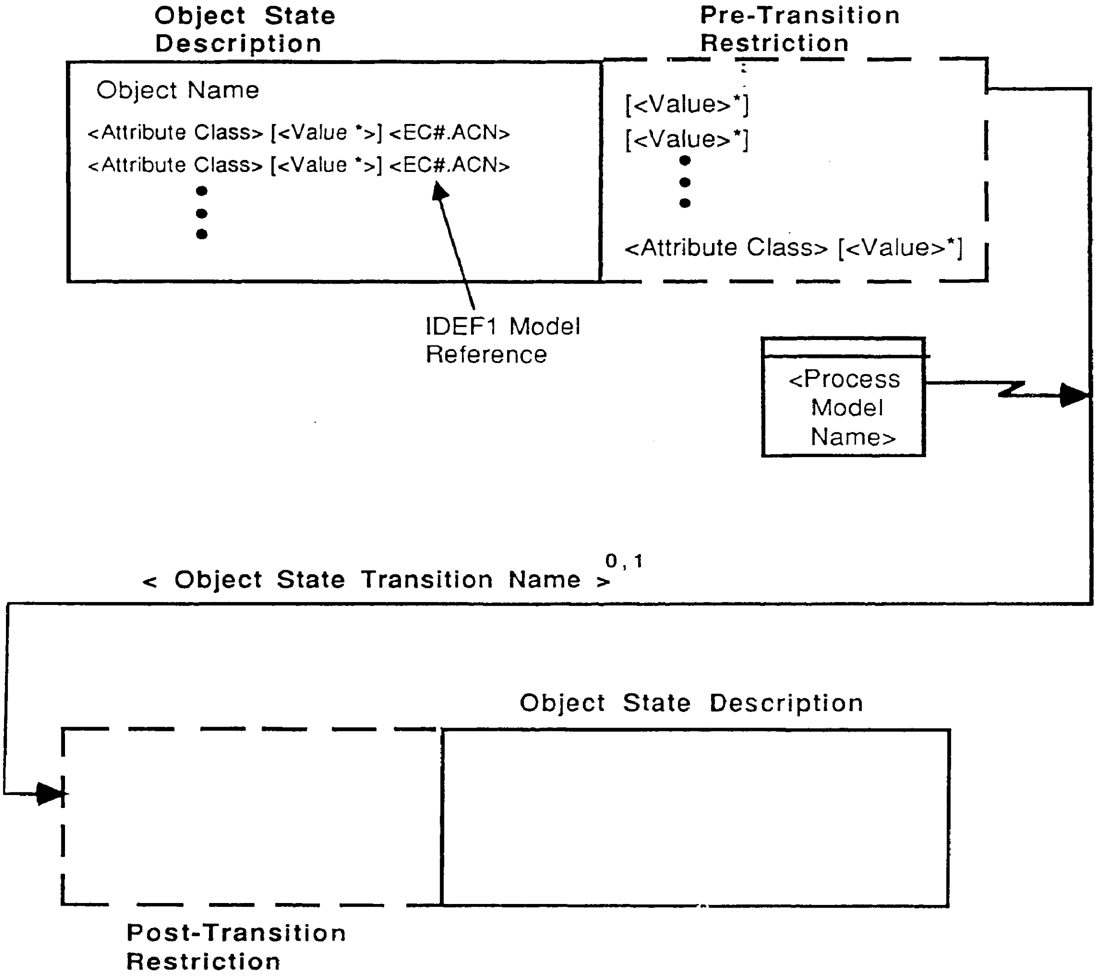

Object State Description
Object Name
<Attribute Class> [<Value *>] <EC#.ACN>
<Attribute Class> [<Value *>] <EC#.ACN>
Pre-Transition Restriction
[<Value>*]
[<Value>*]
<Attribute Class> [<Value>*]
IDEF1 Model Reference
<Process Model Name>
< Object State Transition Name >
0,1
Object State Description
Post-Transition Restriction

FIGURE 8.18: OBJECT STATE TRANSITION SYMBOL SUMMARY.

and expense in the analysis of proposed changes or critical problems. The use of the SDCE in a specific facility over time will result in the establishment of a knowledge base which contains a description of not only the basic objects in that environment (machines, products, tools etc.) and their relationships, but also the "manufacturing logic" which determines how the system works. This logic includes the operating policies of all levels of management in the facility. It also includes the design rational which was determined at the time of the design of the layout of the facility. Figure 8.19 presents an overview of the structure of the SDCE.

The current SDCE was constructed under contract to Chrysler Motors Corporation.[8] In the course of that research several versions, of the knowledge representation component, the description capture component, and the description browsing component were constructed and tested. Initially the idea we pursued was one of having an extensive (but fixed) ontology based on an object hierarchy and specialized editors/browsers for each major concept type. Thus, we designed a "Facility Layout" editor (see Figure 8.20) for capture of the facility descriptions, a "Bill of Materials" editor (see Figure 8.21) for capture of product descriptions, and a "Process" editor for capture of descriptions of the process and associated control logic organized by the responsible organization (see Figure 8.22), and a "Type" editor (see Figure 8.23) for capture of the ontology.

As we attempted to apply these components to the system descriptions which had been gathered, we found that there was no way to control (or accommodate) the fluid nature of an actual description. Simply put, the notion of mere instantiation of a predefined type (or class) was insufficient. As pointed out in Section 5, concepts of the "type" category do not have a fixed set of definitions. However,

[8] The project team included: the author who served as the knowledge engineer and system architect, John Morris who served as the lead designer and programmer, Jackie Wheeler and Sherri Messimer who served as the principle user interface designers, documenters and quality control personnel, Tom Blinn who was responsible for the SDKD implementation, Paul Squiterri who served as the modeler, Dr. Guy Bailey who assisted with the linguistic analysis, Dr. Chris Menzel who assisted in the ontology and knowledge base structure design, and Murali Krishnamurthi who served as a systems consultant.

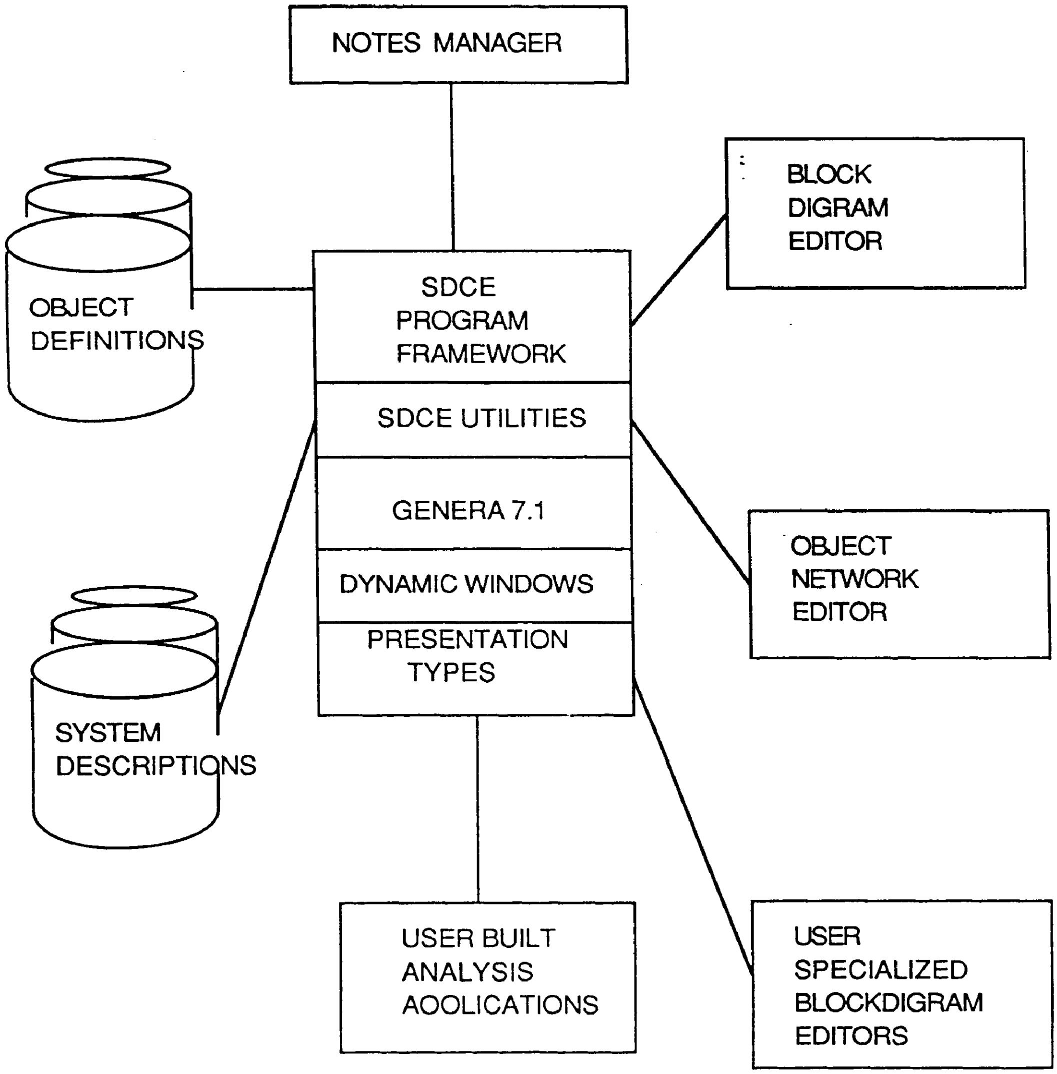
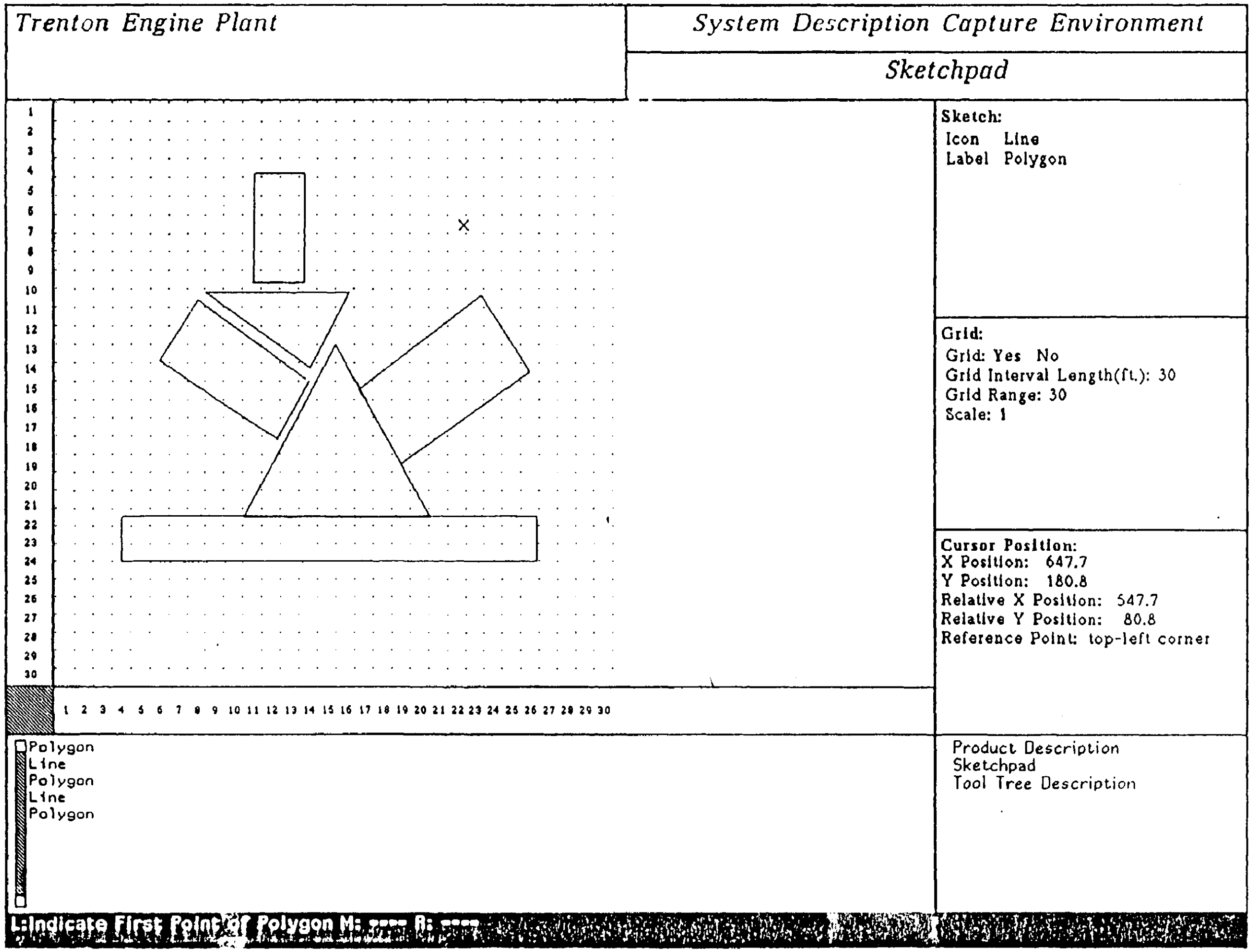


FIGURE 8.19: LOGICAL VIEW OF THE MAJOR COMPONENTS OF THE SDCE.



Trenton Engine Plant

System Description Capture Environment

Sketchpad

Sketch:
Icon Line
Label Polygon

Grid:
Grid: Yes No
Grid Interval Length(ft.): 30
Grid Range: 30
Scale: 1

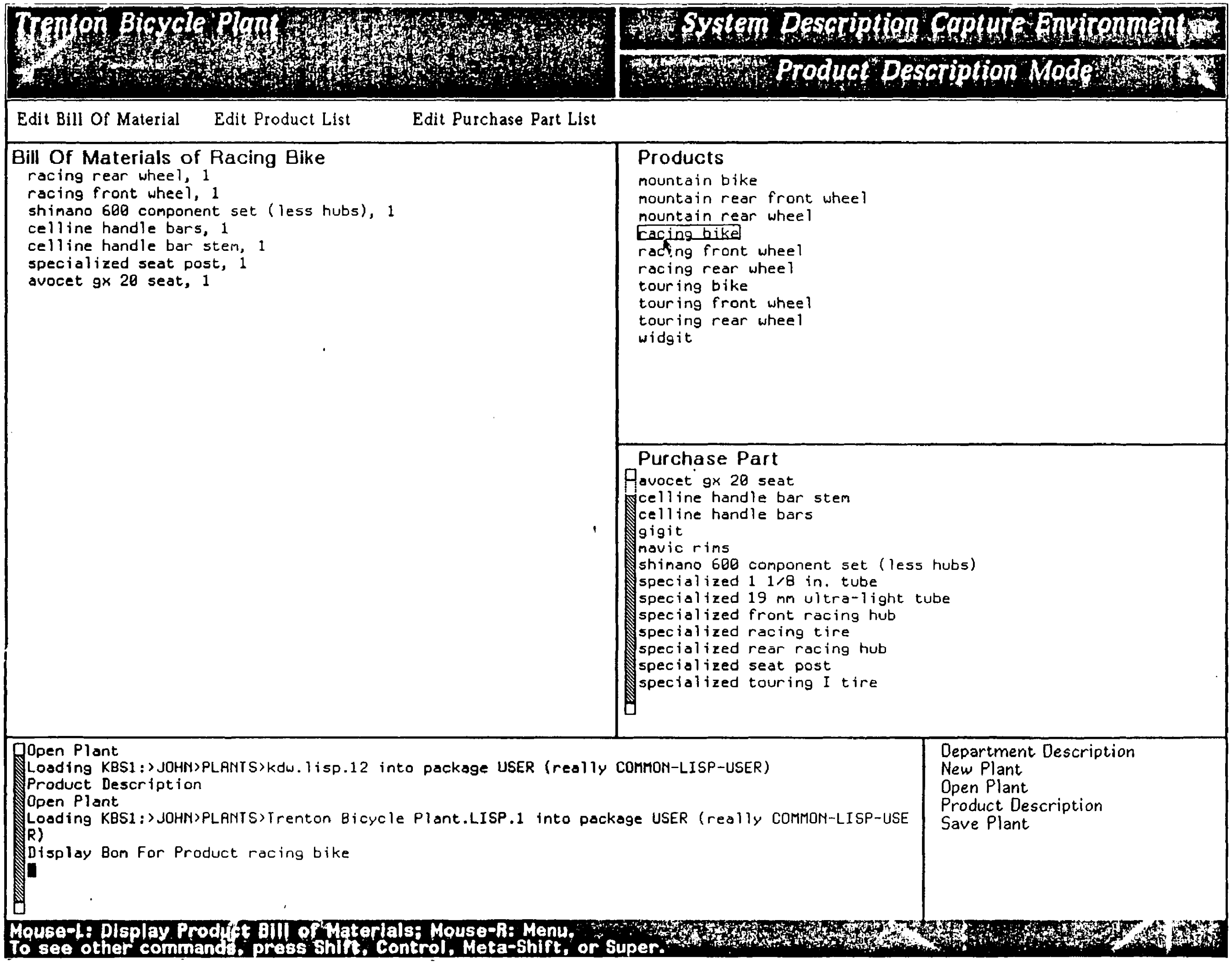
Cursor Position:
X Position: 647.7
Y Position: 180.8
Relative X Position: 547.7
Relative Y Position: 80.8
Reference Point: top-left corner

Polygon
Line
Polygon
Line
Polygon

Product Description
Sketchpad
Tool Tree Description

L:Indicate First Point of Polygon M: ---- R: ----

FIGURE 8.20: SKETCH PAD EDITOR FOR FACILITY DESCRIPTION CAPTURE.



Bill Of Materials of Racing Bike

```
  racing rear wheel, 1
  racing front wheel, 1
  shinano 600 component set (less hubs), 1
  celline handle bars, 1
  celline handle bar sten, 1
  specialized seat post, 1
  avocet gx 20 seat, 1
```

Products

```
mountain bike
mountain rear front wheel
mountain rear wheel
racing bike
racing front wheel
racing rear wheel
touring bike
touring front wheel
touring rear wheel
widgit
```

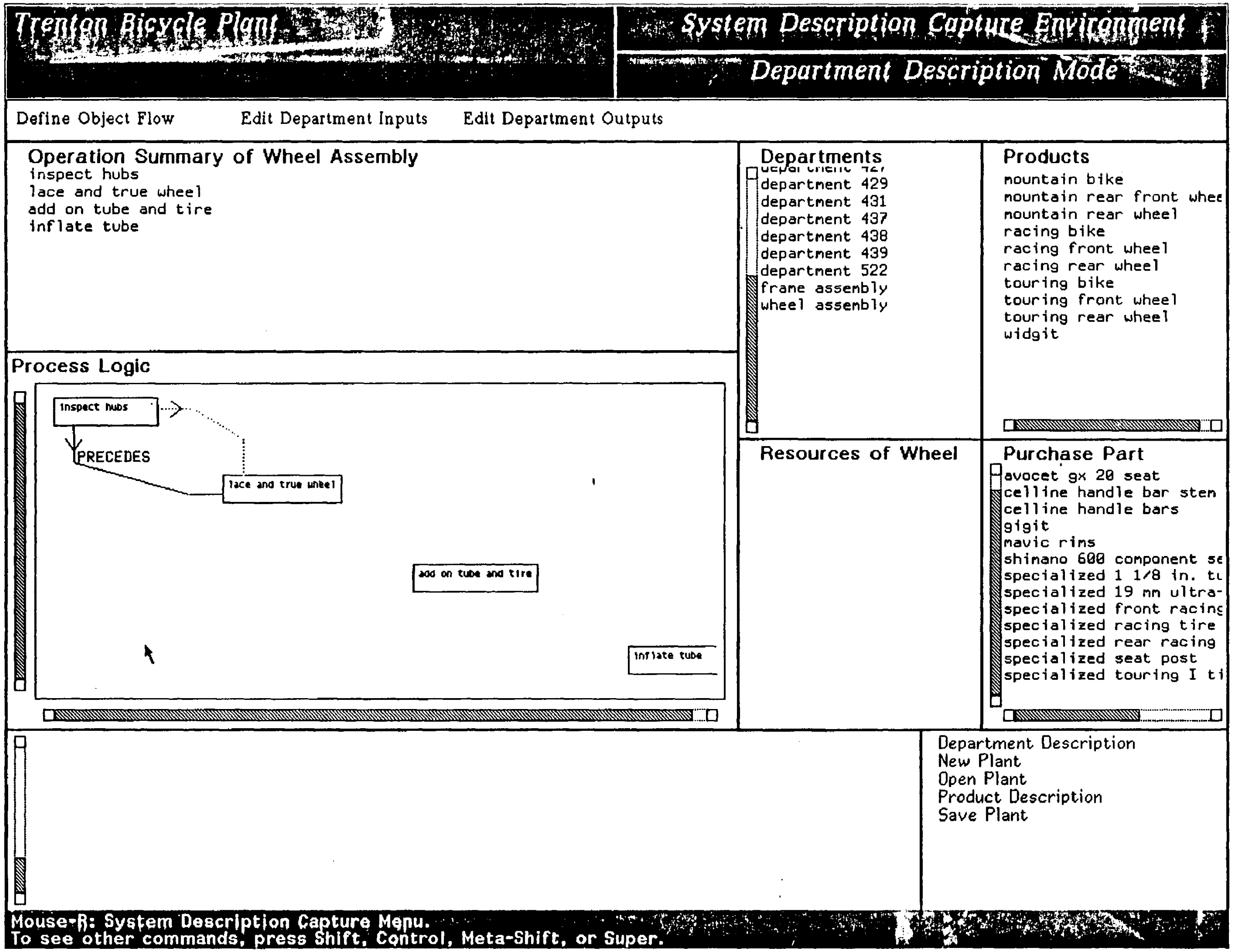
Purchase Part

```
avocet gx 20 seat
celline handle bar sten
celline handle bars
gigit
mavic rins
shinano 600 component set (less hubs)
specialized 1 1/8 in. tube
specialized 19 mm ultra-light tube
specialized front racing hub
specialized racing tire
specialized rear racing hub
specialized seat post
specialized touring I tire
```

```
Open Plant
Loading KBS1:>JOHN>PLANTS>kdw.lisp.12 into package USER (really COMMON-LISP-USER)
Product Description
Open Plant
Loading KBS1:>JOHN>PLANTS>Trenton Bicycle Plant.LISP.1 into package USER (really COMMON-LISP-USE
R)
Display Bom For Product racing bike
```

```
Mouse-L: Display Product Bill of Materials; Mouse-R: Menu.
To see other commands, press Shift, Control, Meta-Shift, or Super.
```

FIGURE 8.21: PRODUCT DESCRIPTION CAPTURE.



Trenton Bicycle Plant

System Description Capture Environment

Department Description Mode

Operation Summary of Wheel Assembly
inspect hubs
lace and true wheel
add on tube and tire
inflate tube

Departments
department 429
department 431
department 437
department 438
department 439
department 522
frame assembly
wheel assembly

Products
mountain bike
mountain rear front whee
mountain rear wheel
racing bike
racing front wheel
racing rear wheel
touring bike
touring front wheel
touring rear wheel
widgit

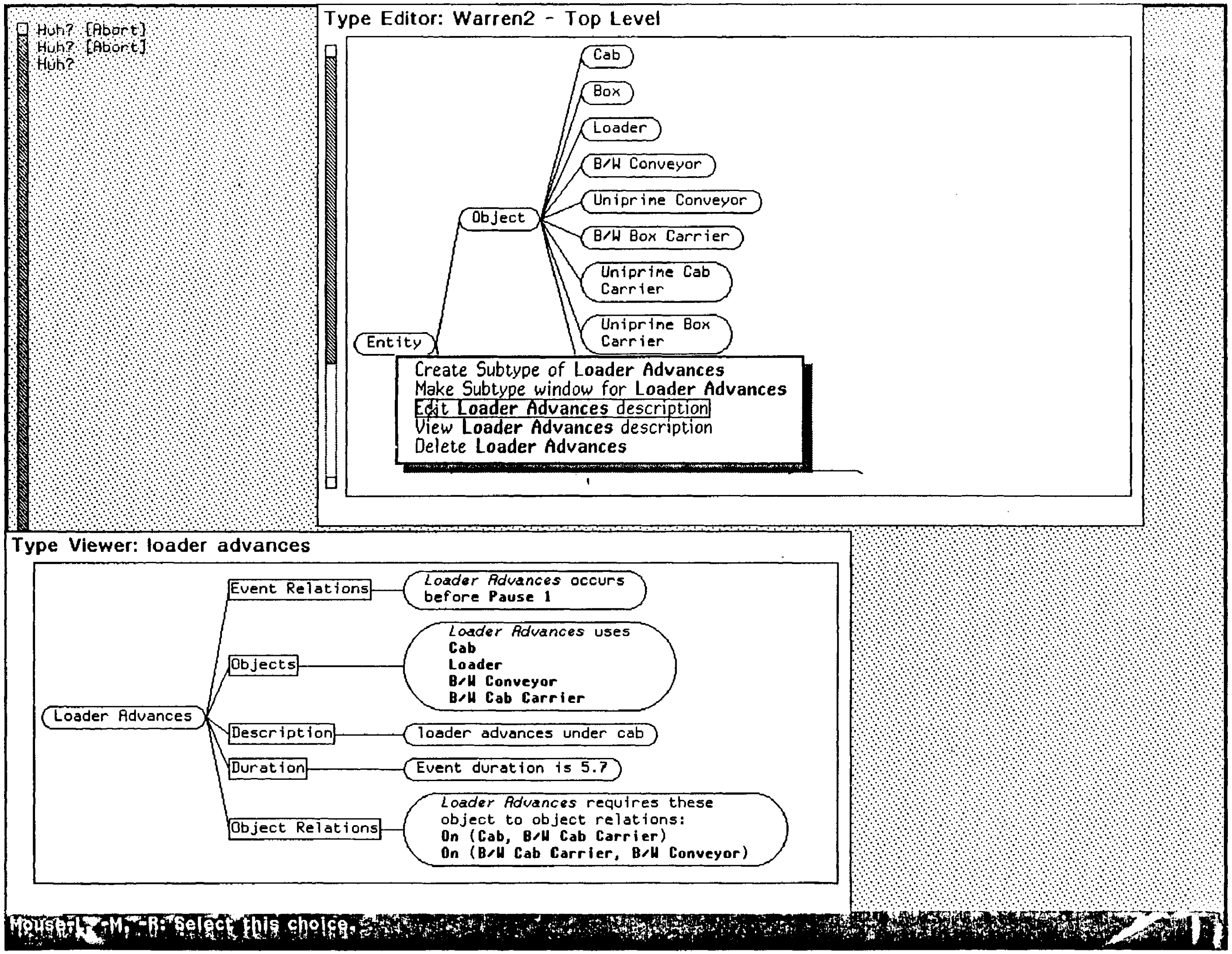
Process Logic

Resources of Wheel

Purchase Part
avocet gx 20 seat
celline handle bar sten
celline handle bars
gigit
mavic rims
shimano 600 component se
specialized 1 1/8 in. tu
specialized 19 mm ultra-
specialized front racing
specialized racing tire
specialized rear racing
specialized seat post
specialized touring I ti

Mouse-R: System Description Capture Menu.
To see other commands, press Shift, Control, Meta-Shift, or Super.

FIGURE 8.22: ORGANIZATION PROCESS DESCRIPTION CAPTURE.



Huh? [Abort]
Huh? [Abort]
Huh?

Type Editor: Warren2 - Top Level

Cab
Box
Loader
B/W Conveyor
Uniprine Conveyor
B/W Box Carrier
Uniprine Cab Carrier
Uniprine Box Carrier
Object
Entity
Create Subtype of Loader Advances
Make Subtype window for Loader Advances
Edit Loader Advances description
View Loader Advances description
Delete Loader Advances

Type Viewer: loader advances

Loader Advances occurs before **Pause 1**

Loader Advances uses
Cab
Loader
B/W Conveyor
B/W Cab Carrier

loader advances under cab

Event duration is 5.7

Loader Advances requires these object to object relations:
On (Cab, B/W Cab Carrier)
On (B/W Cab Carrier, B/W Conveyor)

Mouse-L, -M, -R: Select this choice.

FIGURE 8.23: SDCE TYPE EDITOR.

the problem goes much deeper than this. The process of description formulation appears to be one of:

1. Breaking the subject matter into parts by formulation of names or definite descriptions of the whole and each part,

2. Assigning categories to each part,

3. Naming relations between the parts,

4. Identification of regularities between the parts,

5. Treating each part, relation, or regularity as a piece of subject matter and recursing the process.

Thus, a description of a transfer device might include *The BIW conveyor is an induction motor conveyor with ....* The key is the "with" clause. If one attempts to build a type hierarchy which can accommodate all of the possible completions of this clause one essentially must end up with an infinite hierarchy. In order to address this problem we developed the following capabilities:

1. A block diagram editor (see Figure 8.24) which supports:

   1.1. Declaration of a block category to one defined in the concept editor,

   1.2. Linking one block to another,

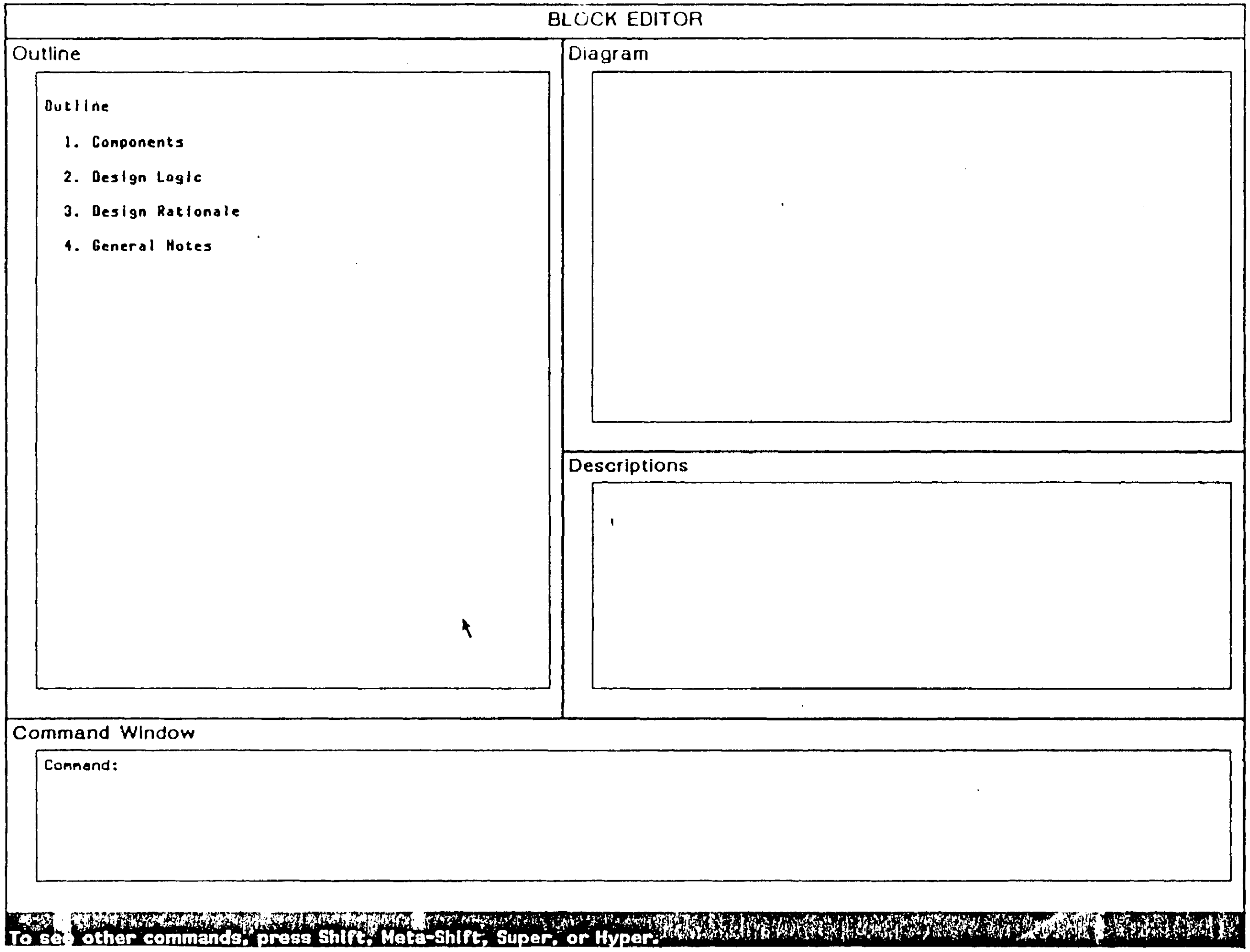
   1.3. Declaration of a link category to one defined in the concept editor,

   1.4. Grouping blocks, links, or blocks and links together into a composite description which may have a category assigned to it,

   1.5. Elaboration of a block, or link, or group with any number of additional block diagrams,

2. Relaxation of the strict hierarchy restriction on the type editor to allow networks of concept structures to be described (see Figure 8.25),

3. Addition of a text annotation capability to handle the capture of arbitrary textual annotations in any mode of the object or block editors.



Outline

1. Conponents

2. Design Logic

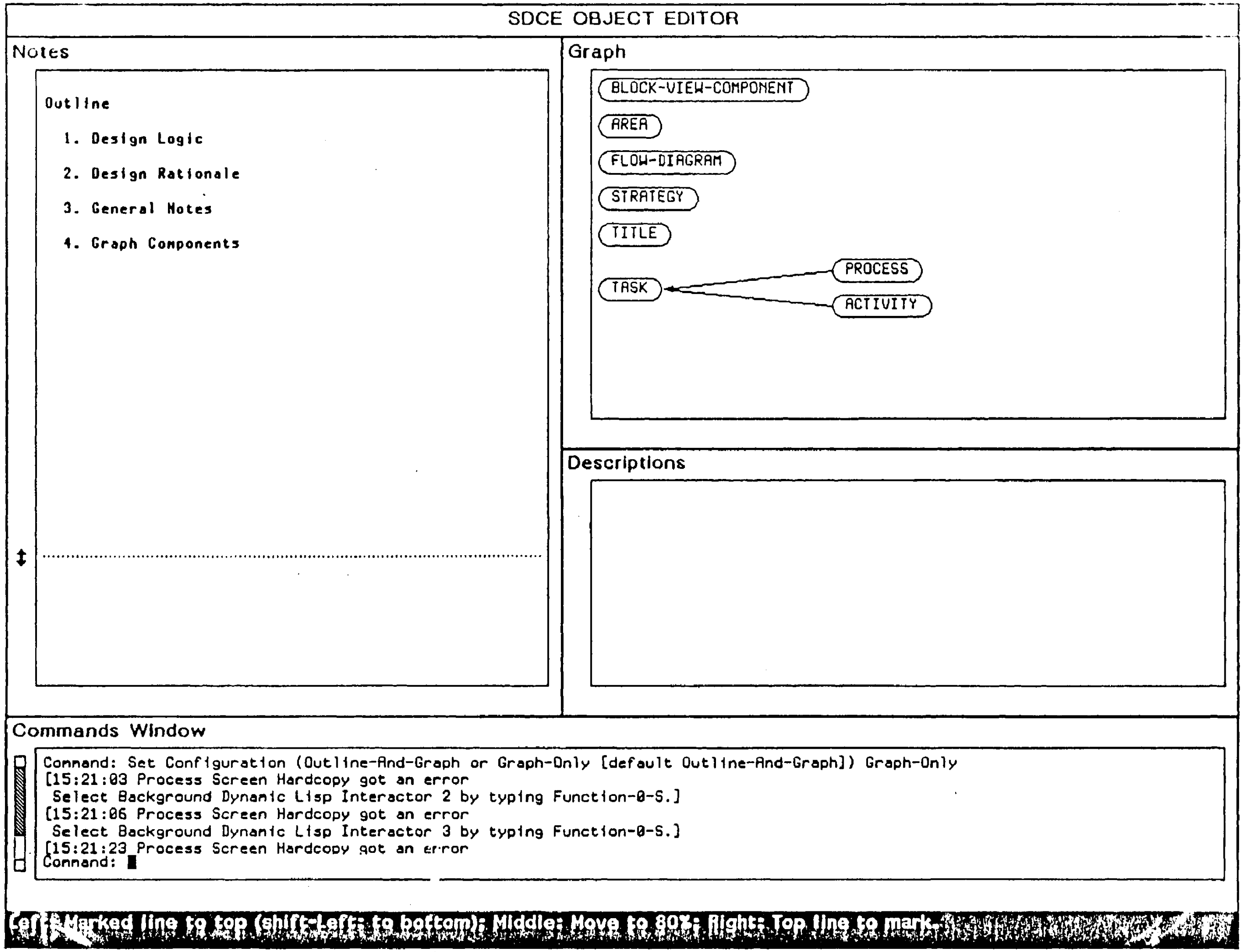
3. Design Rationale

4. General Notes

Connand:

To see other commands, press Shift, Meta-Shift, Super, or Hyper.

FIGURE 8.24: SDCE BLOCK EDITOR.



SDCE OBJECT EDITOR

Notes

Outline

1. Design Logic

2. Design Rationale

3. General Notes

4. Graph Components

Commands Window

```
Connand: Set Configuration (Outline-And-Graph or Graph-Only [default Outline-And-Graph]) Graph-Only
[15:21:03 Process Screen Hardcopy got an error
 Select Background Dynanic Lisp Interactor 2 by typing Function-0-S.]
[15:21:06 Process Screen Hardcopy got an error
 Select Background Dynanic Lisp Interactor 3 by typing Function-0-S.]
[15:21:23 Process Screen Hardcopy got an error
Connand: █
```

Left: Marked line to top (shift-Left: to bottom); Middle: Move to 80%; Right: Top line to mark.

FIGURE 8.25: SDCE OBJECT EDITOR.

The effectiveness of this approach is currently being proven in application within the Simulation and CIM Planning groups at Chrysler Motors. Currently the block editor supports the editing of properties of a block, group, or link. In this mode the user is presented with a menu consisting of the attributes of the type assigned to that block and their default values. The user can modify the default values interactively. Work is currently underway to allow the extension of the attributes of the type assigned to a block from the property editing mode and have the resulting new object type entered as a specialization of the initial object type. This will allow the graceful extension of the terminology component of the system as a natural part of the description capture process.

The SDCE is implemented on the Symbolics lisp machine using the program framework, dynamic window, and presentation type utilities of that system. Initially we implemented the type system within the Flavors substrate. However, in order to support the flexible redefinition of object descriptions and to support the multiple interpretation of those descriptions, we were forced into the design and development of our own SDCE object manager.

8.3 FCT Prototype

The Fact Collection Tool (FCT) was selected for prototyping to determine the feasibility of providing direct automation to the data gathering part of the description process. During this data gathering phase, an analyst is collecting "assertions" made by the domain experts in the environment as well as his own "observations" on the environment. One of the traditional problems has been how to organize and transfer these facts. The FCT prototype represents an approach to providing a micro-based "analyst notebook" which is capable of directly capturing in the form of statements both types of facts. Once captured, the statements can be "interpreted" via classification into a set of semantic categories and further elaborated.

The objective of the FCT concept is to provide a knowledge recording (or experience capture) tool for the analyst to capture and record the facts described by the domain expert or directly observed. This tool must be friendly enough that all information that an interviewed person might give verbally may be typed in without disrupting the train of thought of the speaker. It must also work on hardware that is portable so that it may be taken to the site or to the analyst's workplace.

To achieve these goals, the current prototype FCT was written in 'C' on an IBM PC class machine[9]. In this way it could even be used on inexpensive portables. The user interface design on this system was deemed to be a critical aspect to the successful accomplishment of the design goals. Speed of data entry was considered high on the priority list for this design. Thus, for example, it was determined that such a system would have to be driven by key sequences and not menus in order to perform at a satisfactory speed. The current FCT prototype operates in a stand-alone mode. In the future, it will be integrated with the SDCE in such a way that concept classes can be downloaded and facts collected can be uploaded into the SDCE description data base.

The design used has the flavor of a "semantic net" construction tool in which the user interactively:

1. Captures textual sentences which "represent" an acquired fact,

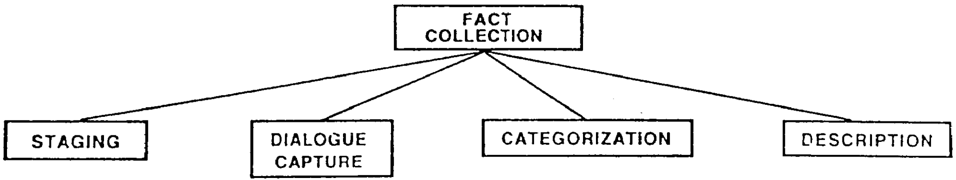
2. Categorizes both the entire text as well as individual words and phrases in the text,

3. Provides further description of the categorized items.

The major modes of the FCT and the features supported by each mode are shown in Figure 8.26.

The main task of the FCT is to accept, store, describe, and relate free form textual statements. When an analyst is in a fact-finding mode, he is acquiring labels (e.g. names, symbols, or descriptive phrases) whose reference is to concepts or objects in the domain of interest as well as complex relations between these

[9] The FCT prototype was developed under contract to the Air Force. The project team included: the author who served as the system architect, William Forsythe and Joel Toland who served as the lead designers and programmers, Mr. Stu Coleman and Dr. Thomas Cullinane who served as technical consultants



FEATURES

• Session Structure - who -where - when • Issues / Objectives • Technique(s) / Session Guide • Notes / Ticklers • Session Chronology • Post-Session Observations	• Dynamic Description Capture • User-Extendable Lexicon • Dynamic Categorization of Dialogue - General - Rule - Relation - Problem	• Dynamic Categorization of Dialogue (Text) Elements - Activity - Physical Object - Artifact - Organization - System - Process / Scenario - Event - Descriptor • Multiple Category Assignment • User-Extendable Categories • Categories Map to Method Requirements	• Template for Each Category • Captures Descriptive Features / Properties • User-Extendable Templates • Features / Properties Map to Method Requirements • Fact Categorization Linkage • Error Tolerance / Correction • Context "Dragging"

FIGURE 8.26: FACT COLLECTION TOOL CHARACTERISTICS.

objects. The reference label can take the form of "names" (e.g. Engineering Department) or descriptive phrases (e.g., the induction motor conveyor in the fabrication department). A pop-up statement capture window is provided for inputting one statement at a time. These statements are assumed to be related, but to contain one idea or piece of information.

The second task of the FCT is to assist the interviewer in organizing his thoughts and questions. To accomplish this, the modeler may classify the statement and any word in the statement as he types. The four statement types are general, rule, relation, or problem statement. The eight main word classifications are activity, physical object, artifact, organization, automated system, process / scenario, descriptor, or event. By classifying a word, that word is placed in a database, and a "tag" associates the entry with the statement from whence it came. At any time the modeler may call up a listing of all entries in a class, or a class specific data form to be filled out about that entry. A ninth class is recall where the modeler marks a word as being of some unknown class when he is not sure which class it should be. In this way, he may look at the entry later and decide where it belongs.

At any point the modeler may change the classification of an entry. The FCT will also accept duplicate entries so as to relieve the burden of remembering what information has already been classified. The modeler may later determine that multiple entries are the same and combine them into a single entry retaining all of the tags to the original statements. These tags, along with time stamps, will help the modeler to reconstruct the meeting in order to help him determine the exact meaning of an entry by viewing the context in which the entry was used.

An example scenario of dialog capture is displayed in Figure 8.27. This figure illustrates the general process of statement capture, label / phrase classification. Figure 8.28 displays the 10 classification categories provided by the prototype FCT. Figure 8.29 displays a description capture form for a label whose referent has been classified as being of type "Physical Object".

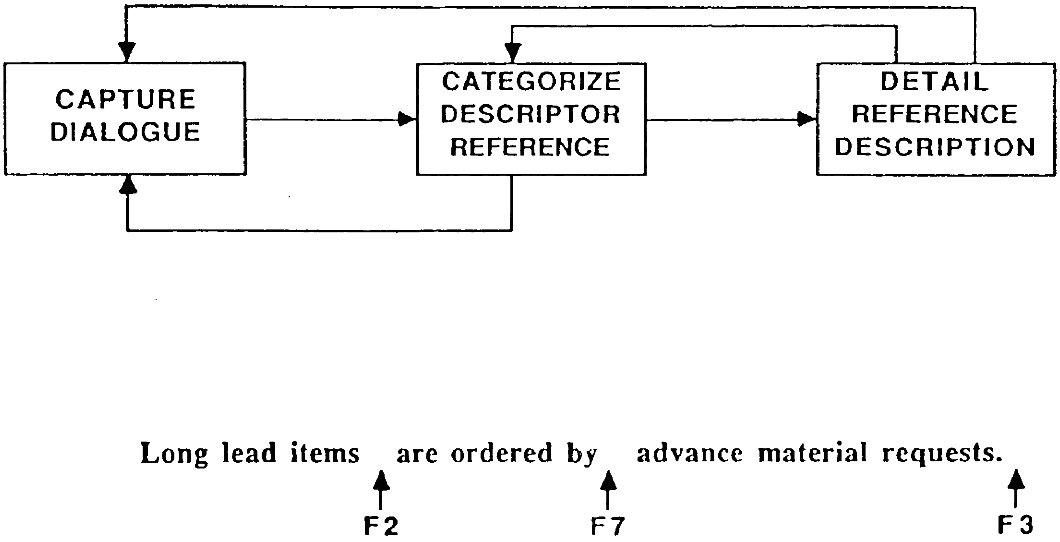
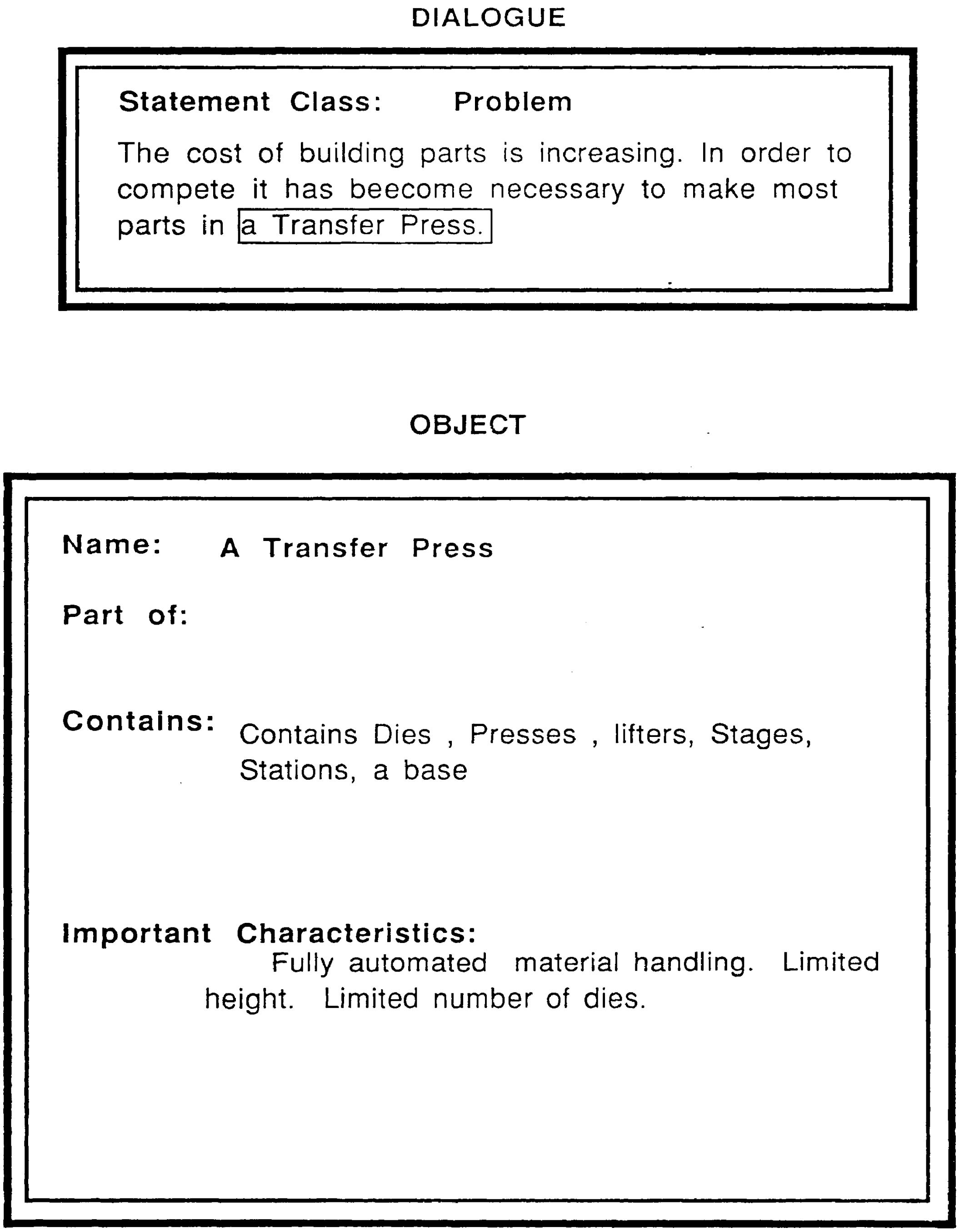

CAPTURE DIALOGUE
CATEGORIZE DESCRIPTOR REFERENCE
DETAIL REFERENCE DESCRIPTION
Long lead items are ordered by advance material requests.
F2
F7
F3

FIGURE 8.27: DIALOG CAPTURE USING FCT.

Activity
Physical Object
Artifact
Organization
System
Process/Scenario
Event
Link
Descriptor
Statement

FIGURE 8.28: FCT CLASSIFICATION SCHEMES.



DIALOGUE

Statement Class: **Problem**

The cost of building parts is increasing. In order to compete it has beecome necessary to make most parts in a Transfer Press.

OBJECT

Name: **A Transfer Press**

Part of:

Contains: Contains Dies , Presses , lifters, Stages, Stations, a base

Important Characteristics: Fully automated material handling. Limited height. Limited number of dies.

FIGURE 8.29: FCT DESCRIPTION FORM FOR PHYSICAL OBJECT TYPES.

8.4 MDSE Prototypes

The model development support environment efforts spanned several projects. The following describes initiatives in each of the modeling areas described in Section 3.

8.4.1 Implementation Model Support

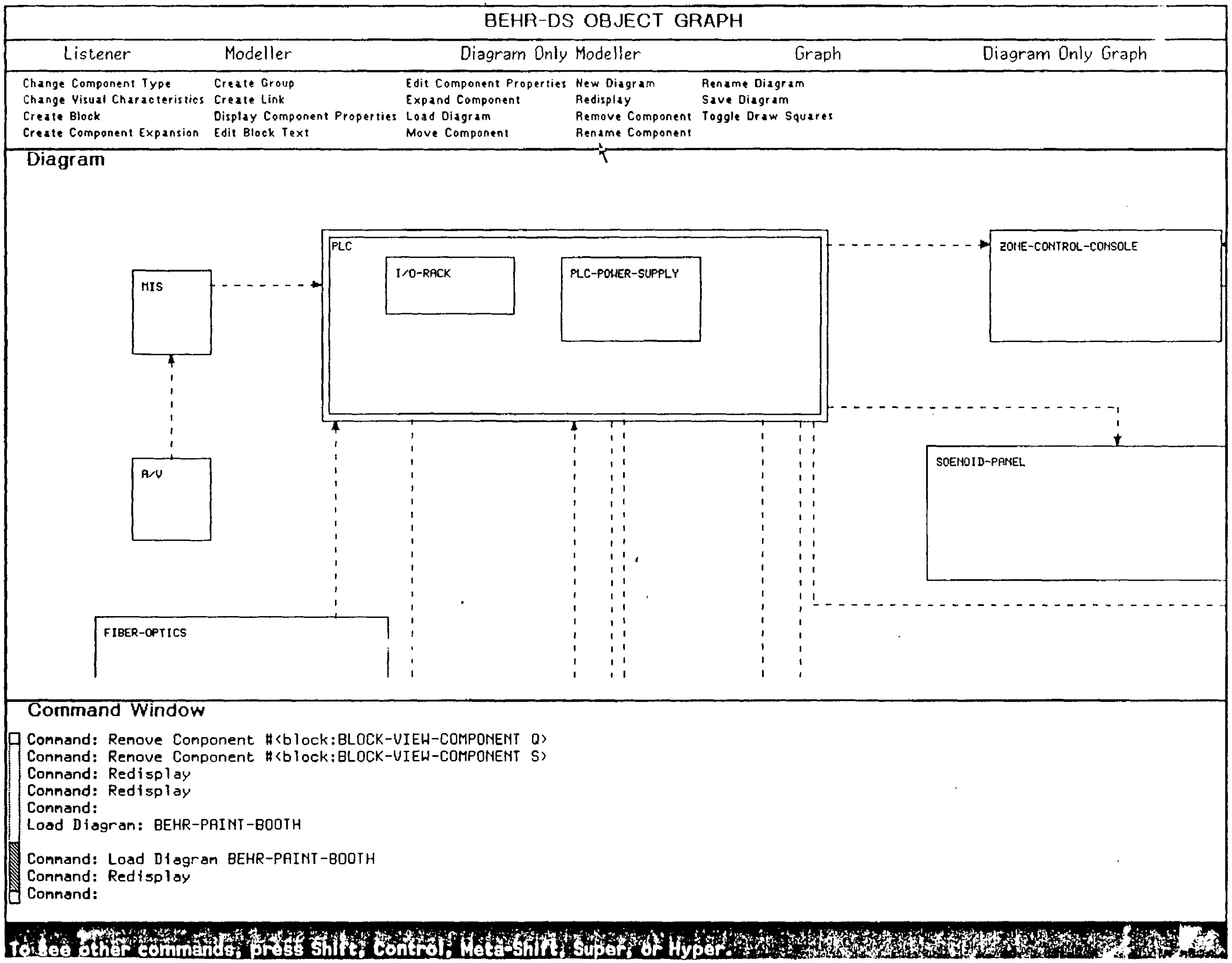
One of the requirements identified in Section 3 which the KAMSS must support is the rapid prototyping of engineering decision applications (implementation models). The SDCE has been used as a platform for providing the initial capabilities of an Engineers Programmers Workbench.

By moving away from specialized editors to the block diagram editor concept (which is specialized by the types assigned to the blocks), the current SDCE can be used to capture descriptions of software designs. In fact the object editor provides an option to save the object network as Flavor declarations which provides a means for engineers to rapidly develop applications using object programming.

The block diagram editor can also be used to capture descriptions of situations for applications other than factory and software descriptions. It is currently being used as a means of capturing device descriptions for a causal model-based diagnosis application (see Figure 8.30). In this application the user who is describing a piece of equipment (or a system) is restricted to the object types used by the causal model based reasoner.

8.4.2 Simulation Model Design Support

A simulation model design support system is being built on top of the SDCE prototype using the block diagam editor and object editor to support the design of simulation models based on the SIMAN simulation language constructs. This system provides much of the same capabilities as were originally prototyped in the OBMODELER described in the next subsection of this section. Using the SDCE Object editor, a type hierarchy was defined of the SIMAN modeling concepts. The model designer starts with a block diagram description of the system to be modeled and overlays the groups (or blocks) which represent the abstraction or elaboration of the system description into a model design. The modeler can



```
Connand: Renove Conponent #<block:BLOCK-VIEW-COMPONENT Q>
Connand: Renove Conponent #<block:BLOCK-VIEW-COMPONENT S>
Connand: Redisplay
Connand: Redisplay
Connand:
Load Diagran: BEHR-PAINT-BOOTH

Connand: Load Diagran BEHR-PAINT-BOOTH
Connand: Redisplay
Connand:
```

FIGURE 8.30: SDCE BASED MECHANISM DESCRIPTION CAPTURE TOOL FOR DIAGNOSIS.

examine the system characteristics interactively during the specification of the SIMAN model element characteristics. This allows the simulation analyst and the domain expert to operate independently using the SDCE as the vehicle for communication. The domain expert can easily see what parts of the system are being modeled and examine the abstractions made by the modeler.

An analyzer is under development which will accept this graphic simulation design and produce the SIMAN code for each of the components specified in the model. As experience is gained in the use of this support mode of operation, the results of Section 7 will be used to develop more intelligent assistance for the model design process.

8.4.3 IDEF Model Development Support

The Integrated Model Development Support Environment is a suite of integrated software tools that provide intelligent support for system modeling. This prototype was constructed as a proof of concept demonstration of the MDS component of the KAMSS. The initial prototypes have focused on support for the IDEF0 and IDEF1 modeling and on the construction of a generalized tool for building such modeler tools called the "MetaModeler." The IDEF tools assist in the complete process of function and information modeling[10]. The current IDEF model building system is only loosely integrated with the SDCE (callable as an option from the SDCE framework).

[10] The MDSE prototype was developed under contract to the Air Force. The project team included: the author who served as the system architect, Mr. Tom Sheppard (Symbolics Inc.) who served as the lead designer and programmer of the MetaModeler and the IDEF1 modeling tool, Al Underbrink and Martha Wells who served as the principle designers and programmers for the discription collection and data dictionary components, Louis Decker and Keith Ackley who served as the lead designers of the IDEF0 modeling tool, Paula Mayer and Charles Bodenmiller who developed the English language model summary generator, Steve Cook who developed the report generation and hardcopy output utilities, Terre Layton responsible for testing and documentation, Mr. Stu Coleman and Dr. Thomas Cullinane who served as technical consultants

An effective modeler support environment must also support the integration of a large number of models being developed either by the same modeler or different modelers. It must also support reuse of existing models. A great deal of the work done by a modeler involves the integration of a model or a part of a model into another model. This integration is similar to the merging of two models to form a third. The main difference between this and the creation of a model is that in these cases the model elements already exist.

The design of the IDEF0 modeler is based upon the MetaModeler concept. The MetaModeler allows a description of the information to be entered and generates the basic functionality necessary for a graphics-oriented tool. An IDEF1 model of IDEF0 was developed to use as input to the MetaModeler. The IDEF0 model developed is shown in Figure 8.31. An appropriate description was created for the MetaModeler, supplied to the MetaModeler, and an IDEF0 modeling tool was generated. Figure 8.32 displays the single screen version of the IDEF0 model builder.

A major portion of the development of the prototype IDEF0 modeler was the user interface. The command interpreter, mouse gesture handler, and graphic display were design and constructed to complete the IDEF0 tool. Much of the functionality was intentionally designed to run concurrently with the IDEF1 modeling tool. The user has the ability to operate on models of either modeling methodology with each loaded into the AutoIDEF system at the same time.

As the IDEF0 tool is integrated with the IDEF1 tool through the MetaModeling framework, it is possible for the modeler to build such models simultaneously pulling concepts from one to the other. Figure 8.33 illustrates such a dual model session.

The modeling support for IDEF1 was also constructed using the MetaModeler generation capabilities. The IDEF1 model of IDEF1 which was used as input to this generation feature is shown in Figure 8.34. Figure 8.35 displays a sample screen from the IDEF1 modeler.

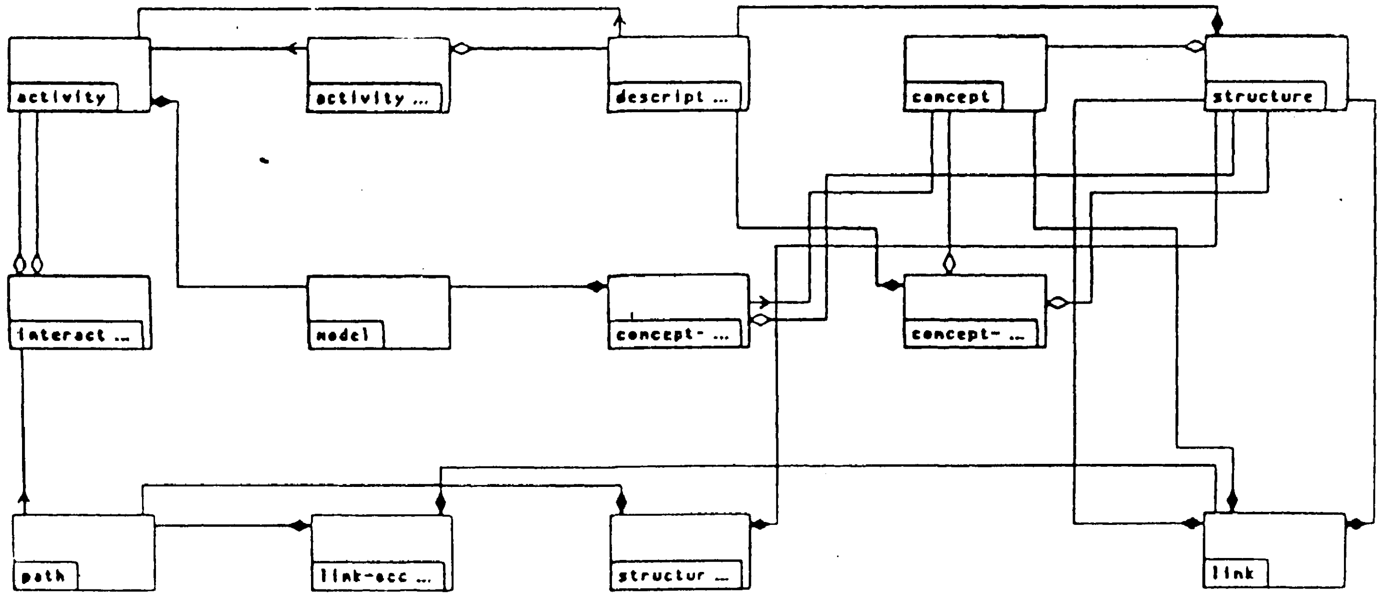
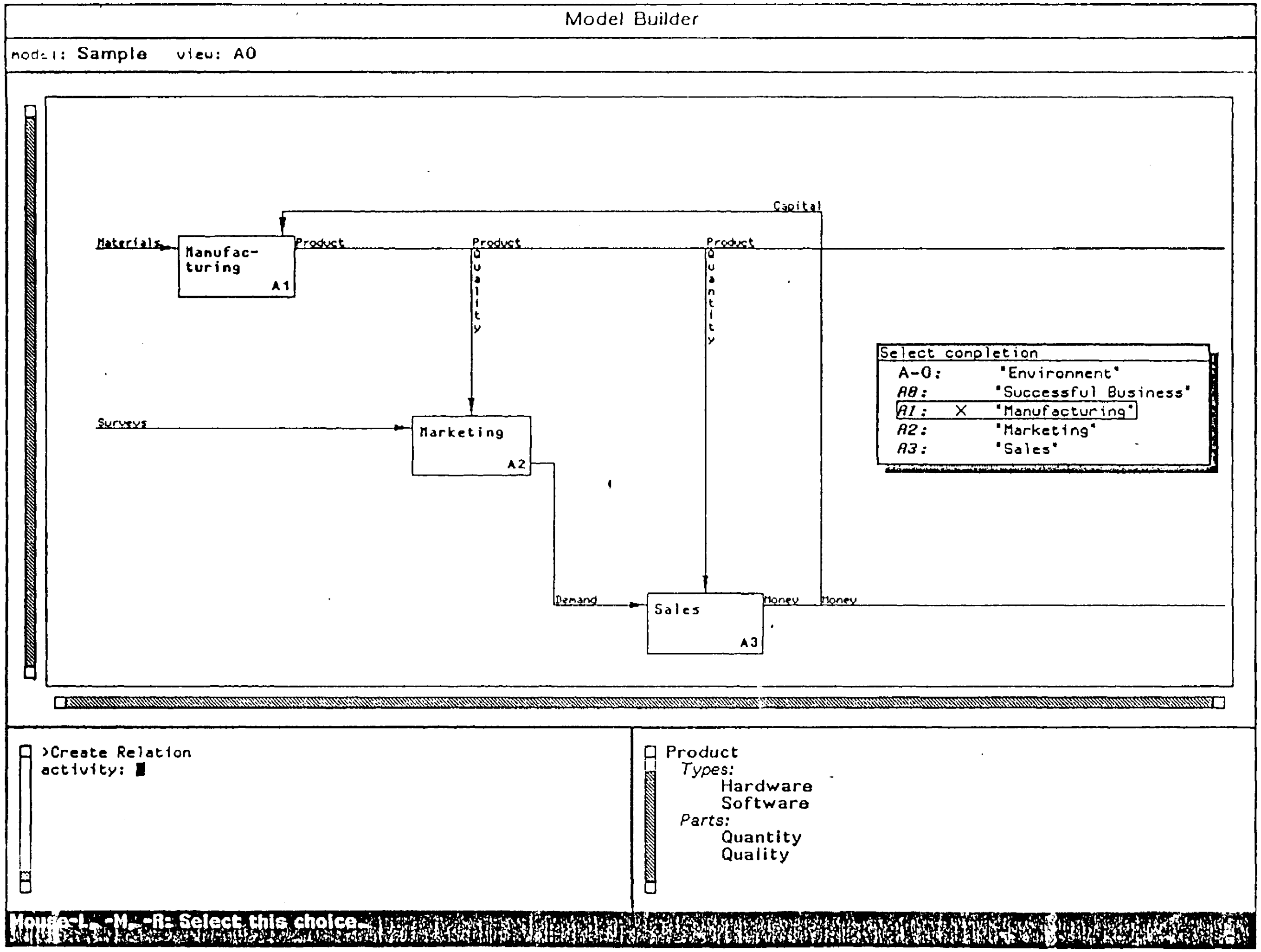


FIGURE 8.31: IDEF1 REPRESENTATION OF IDEF0.



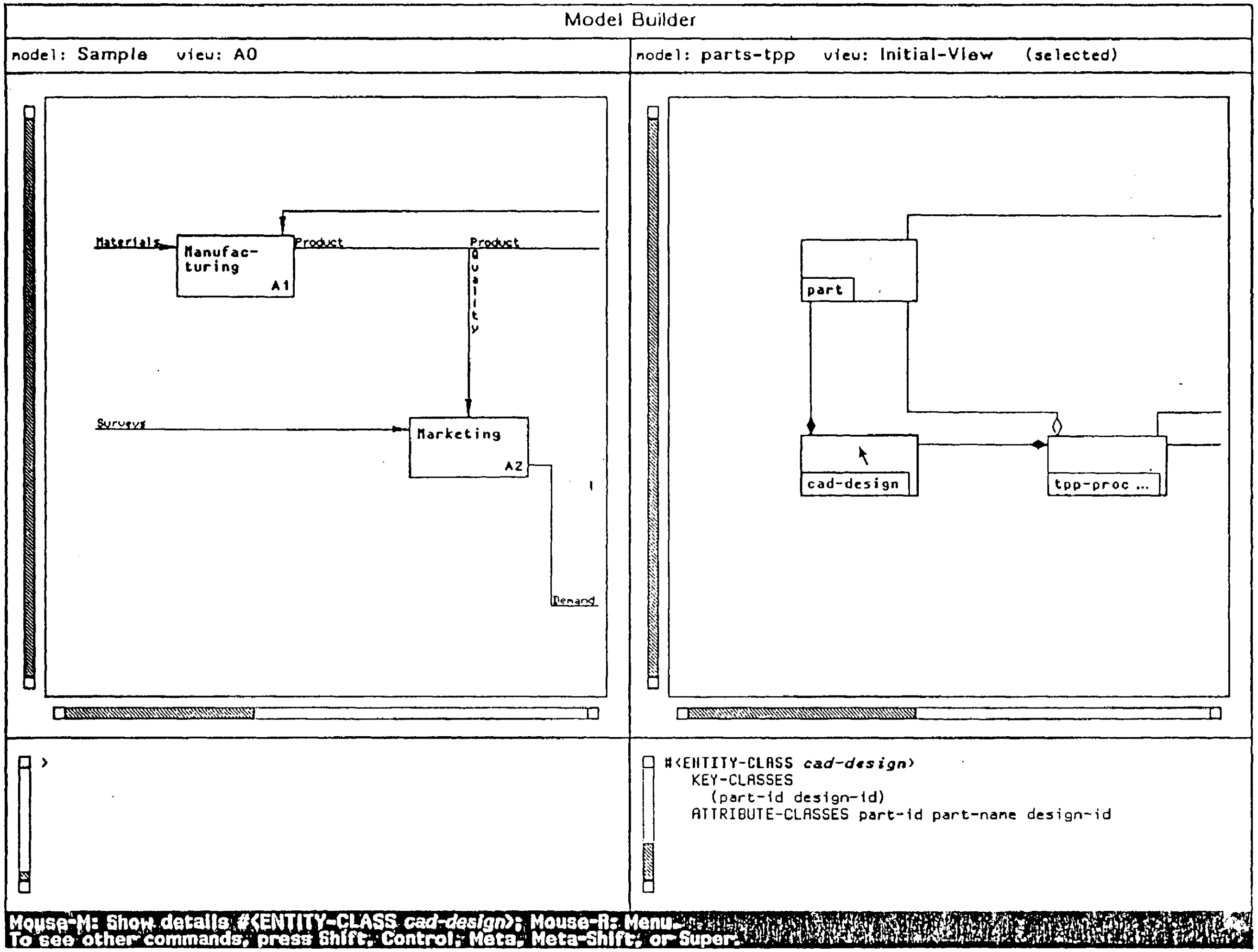
Select completion

A-0:		'Environment'
A0:		'Successful Business'
A1:	X	'Manufacturing'
A2:		'Marketing'
A3:		'Sales'

```
>Create Relation
activity: █
```

Product
Types:
Hardware
Software
Parts:
Quantity
Quality

FIGURE 8.32: IDEF0 MODELING SUPPORT SCREEN.



```
#<ENTITY-CLASS cad-design>
  KEY-CLASSES
    (part-id design-id)
  ATTRIBUTE-CLASSES part-id part-name design-id
```

```
Mouse-M: Show details #<ENTITY-CLASS cad-design>; Mouse-R: Menu.
To see other commands, press Shift, Control, Meta, Meta-Shift, or Super.
```

FIGURE 8.33: MULTIPLE MODELING SUPPORT.

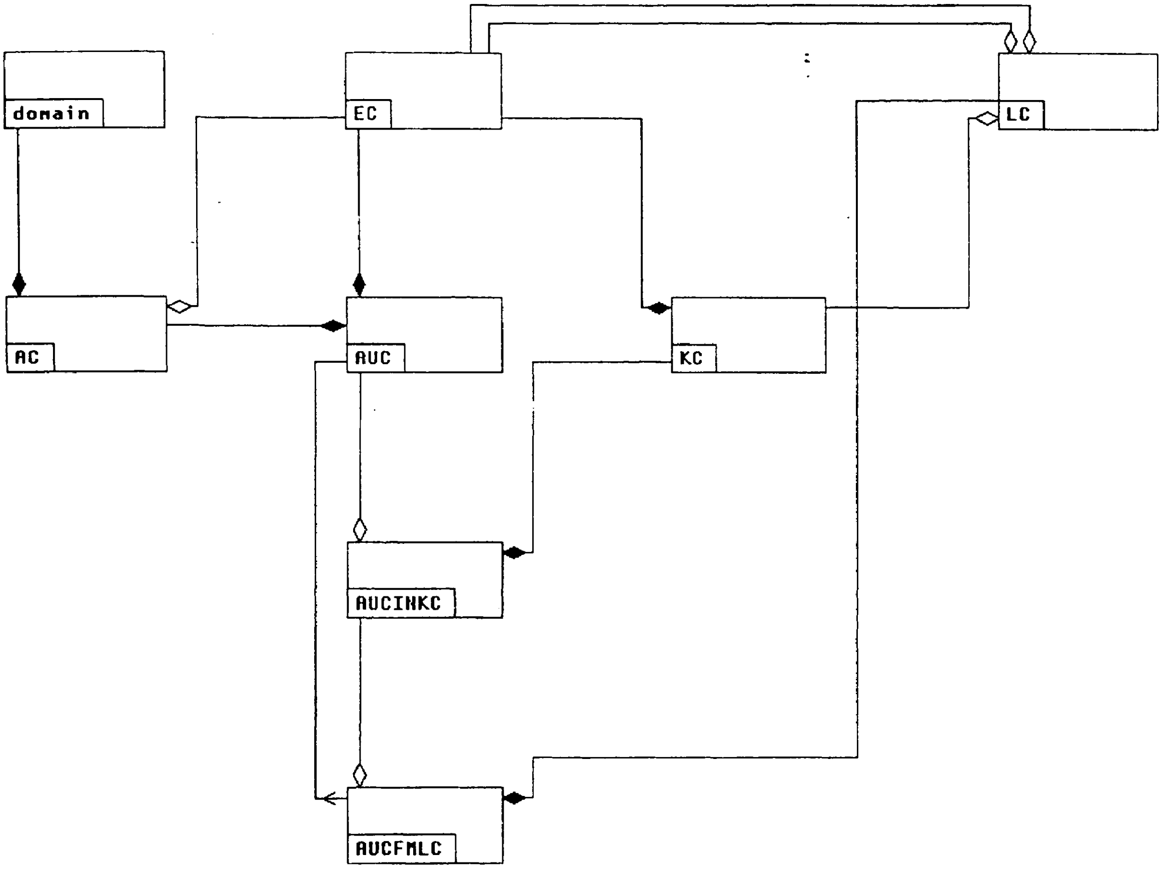
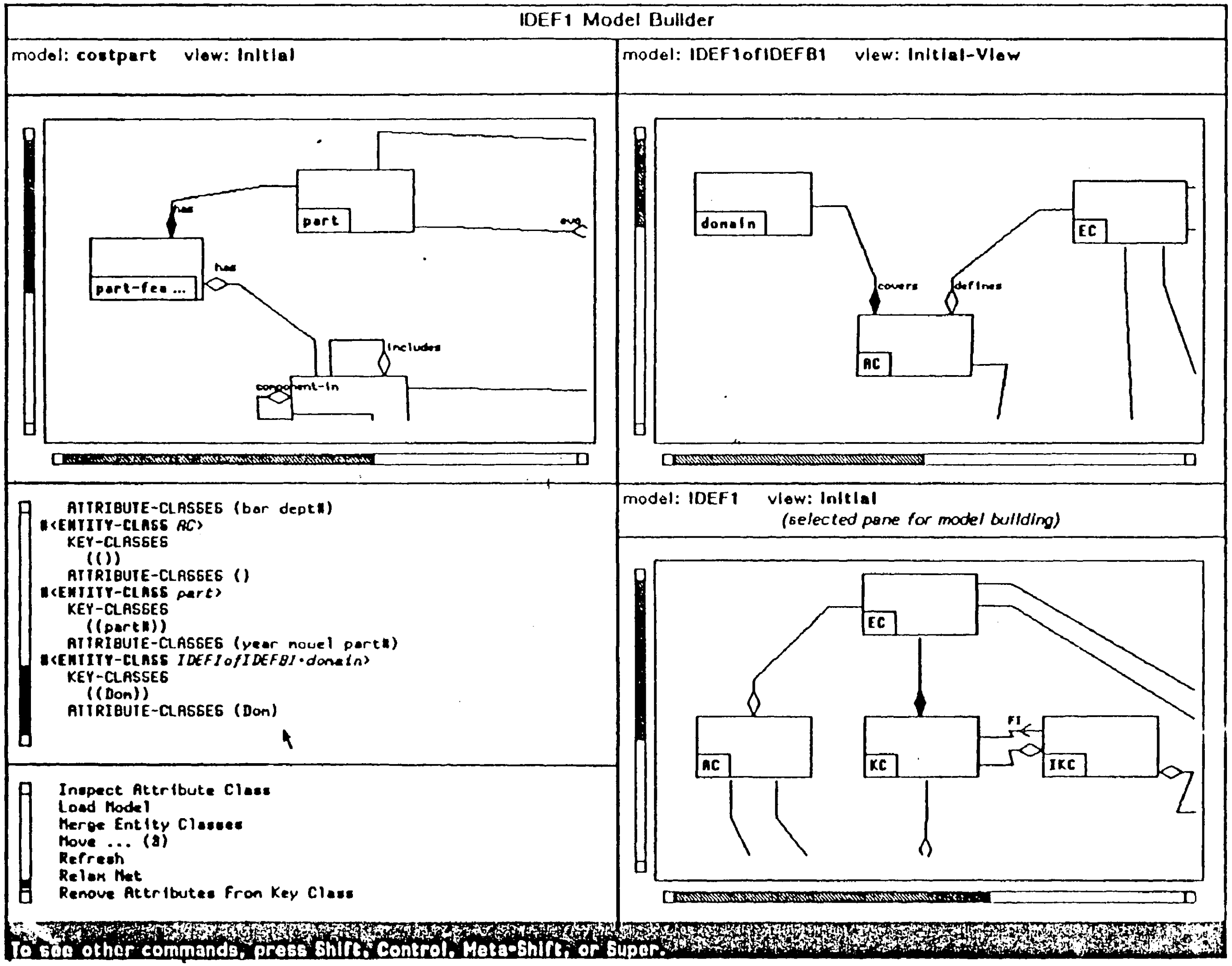


FIGURE 8.34: IDEF1 REPRESENTATION OF IDEF1.



IDEF1 Model Builder
model: costpart view: Initial
part
has
avg
part-fea ...
has
includes
component-in
model: IDEF1ofIDEFB1 view: Initial-View
domain
covers
defines
EC
AC
model: IDEF1 view: Initial
(selected pane for model building)
EC
AC
KC
FI
IKC

```
   ATTRIBUTE-CLASSES (bar dept#)
#<ENTITY-CLASS AC>
   KEY-CLASSES
     (())
   ATTRIBUTE-CLASSES ()
#<ENTITY-CLASS part>
   KEY-CLASSES
     ((part#))
   ATTRIBUTE-CLASSES (year novel part#)
#<ENTITY-CLASS IDEF1ofIDEFB1·domain>
   KEY-CLASSES
     ((Don))
   ATTRIBUTE-CLASSES (Don)
```

To see other commands, press Shift, Control, Meta-Shift, or Super.

FIGURE 8.35: IDEF1 MODEL BUILDER SUPPORT INTERFACE.

8.4.4 Generalized Model Generator

The MetaModeler effort has focused on development of an object based code generation system which would accept formal descriptions of a methodology and produce an intelligent modeling support tool from that description. The current version of the MetaModeler generates a basic model data management tool to which the developer must add the user interface (see Figure 8.36).

The MetaModeler provides a general way to create a computerized tool for any modeling methodology. It provides a specification language that allows the description of the syntax and grammar elements of modeling methodology. The first step in the use of the MetaModeling tool is to describe the information structure and semantics of the methodology for which a modeler is to be designed. Using IDEF1, the programmer describes the entity classes and link classes in the specific MetaModeler language. The third step is to let the MetaModeler generate the model tool. The last step in the process is to add the user interface. Currently only the graphical display object management and link routing of this last step is automatic. The code generated by the MetaModeler consists of a set of Flavors object definitions and Lisp methods.

The current MetaModeler was used in the development of the IDEF1 and IDEF0 modeling tools. The value of the MetaModel concept was proven in the development of these modeling tool in terms of development time savings and integratability of the resulting systems. The concepts behind the MetaModeler are still undergoing an evolutionary process and continue to change as more experience is gained with its use.

8.5 OBSIM: An Object Based Simulation Language

The objective of this language development was to establish a framework for the development and evaluation of prototype simulation languages for the model generator component and the simulation engine component of the KAMSS. The current implementation is being used to investigate the following three simulation concepts:

1. Separation of the user supplied system description from the design of the simulation model.

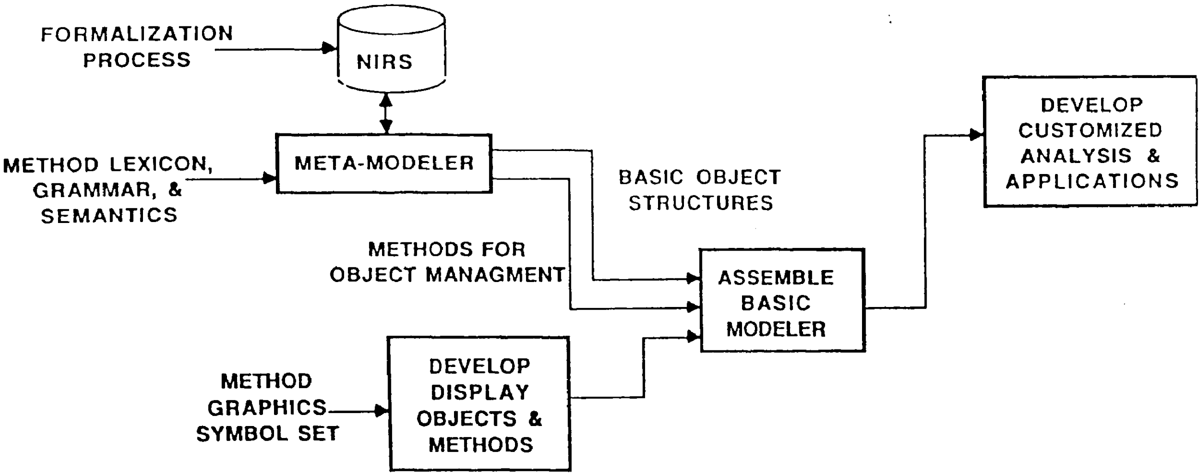

FORMALIZATION PROCESS
NIRS
METHOD LEXICON, GRAMMAR, & SEMANTICS
META-MODELER
BASIC OBJECT STRUCTURES
METHODS FOR OBJECT MANAGMENT
ASSEMBLE BASIC MODELER
DEVELOP CUSTOMIZED ANALYSIS & APPLICATIONS
METHOD GRAPHICS SYMBOL SET
DEVELOP DISPLAY OBJECTS & METHODS

FIGURE 8.36: USE OF METAMODELER TO GENERATE MODELING SUPPORT ENVIRONMENTS.

2. Provision of both entity tracing and condition triggering types of simulation modeling support in the same environment.

3. Use of an electronic blackboard concept for system definition and simulation model design support.

OBSIM is currently implemented in the LOOPS [Bobrow and Stefik 1983] language on a XEROX 1108 AI workstation with low level functions programmed in InterLisp [Kaisler 1986].

As described previously in this dissertation, a simulation analysis process is initiated with the definition of the customer's goals for analysis and system description. One of the problems inherent in existing simulation support environments is that the languages which are provided for these environments only support the specification of the simulation model of the system. Thus, while very powerful aids are provided by the system, they tend only to support the needs of the simulation analyst and not those of the end customer. To understand the representations constructed in these tools requires that the customer be trained in the techniques of both simulation modeling and the particular underlying language constructs. In OBSIM, we have attempted to provide support for both the system description and the translation of that description into a simulation model. Using an object based system description language similar to GEIST [Balzar and Goldman 1982], the customer's description can be recorded and displayed to whatever level of detail is required to satisfy the user. Rather than use the formal axiomatic procedural description language common to most system specification languages we have adopted the rule based language similar to the rule language in LOOPS.

The simulation model design proceeds by the assignment of model objects to system description objects or groups of objects. The simulation analyst can choose from a primitive set of OBSIM model objects or he can utilize modeling objects from existing simulation languages. In the analysis of what knowledge an expert simulation analyst possesses about the design of simulation models, we dicovered that much of that knowledge has been encapsulated into the modeling constructs which are available in existing simulation languages. Therefore in the OBSIM language, we have taken advantage of the inheritance properties of the LOOPS object programming capabilities to implement equivalent modeling constructs. Thus, an analyst can use SLAM activities or GPSS delay blocks [Schriber 1972] intermingled with his own customed designed modeling constructs. This approach differs from other object based simulation languages where the simulation construct set is essentially fixed [IntelliCorp 1985b].

8.5.1 Entity Tracing Versus Condition Triggering

One of the standard ways of characterizing discrete event simulation models is the distinction by event, activity, or process orientation. However, in examining the types of analysis which are performed using these modeling approaches, there appears to be two categories of analysis approaches. The distinction is based on what is being observed about the system under study. In traditional manufacturing studies, the primary orientation is focused on the tracing of items through a series of processes. In the man-machine, software architecture, and control logic simulations, the orientation is primarily focused on observing the triggering sequence determined by condition satisfaction and resulting in the actions taken by the system. The goals of the entity tracing type simulations normally focus on prediction of resource contention and overall system performance. The goals of the condition triggering type of simulation normally focus on fault analysis or the determination of the cause of the conditions which gave rise to a particular behavior. When analyzing systems using the entity tracing approach, alternatives are expressed by changing levels of resources, number of entities in the system, changing the characteristics of a process, or changing the flow of entities through the processes. When analyzing systems using the condition / triggering paradigm, the design issues at hand normally dictate the changing of the logical structure of the system or the rules which govern the system operation.

The simulation languages which have evolved to support these different analysis/usage paradigms have focused (appropriately so) on making it easy to change those components which are frequently changed. Thus, for example, changing a single parameter on a branch node in a network language can drastically change the flow of items through the system. However, as today's manufacturing situations demand examination of the integration aspects of men, machines, decision logic, and software control (particularily in evaluation of CIM applications) the need to integrate these two paradigms is beginning to surface. OB-

SIM supports the integration of both kinds of analysis by allowing the specification of entity flow through the traditional network stuctures and the representation of conditional logic through LOOPS rule sets which can exist as stand-alone "logic" nodes or be attached to another standard modeling concept as a special purpose method. Thus, for example, experimenting with different control logic and structures in an FMS situation is as easy as the changing of the English-like IF...THEN rules associated with the supervisory control object.

8.5.2 OBSIM Language Constructs

The OBSIM language consists of four basic classes of objects. The core set of objects and their interaction is displayed in Figure 8.37. System description objects are used in the construction of the user's definition of his system and his goals for analysis. The modeling class of objects contains the primitive dynamics representation objects and the higher level simulation primitives built as mixins of these primitives. The simulator objects are those required to actually simulate a model design (e.g., Supervisor, Clock, Statistics Collector, Deviate generator, etc.). The fourth class of objects are those which provide the development support for the definition and analysis project. These objects are structured around the typical life cycle artifacts associated with a simulation analysis (e.g., runs, models, system definitions etc.). The entire OBSIM language has been implemented in LOOPS objects and rules. Thus, the user of OBSIM or a researcher interested in examining alternative language constructs, can easily modify the behavior of the OBSIM system by redefining the basic object types or by redefining the behavior of the existing objects by modification of the rules which are used to specify the methods which prescribe that behavior. Figure 8.38 displays the structure of a typical rule method attached to an OBSIM object.

OBSIM was originally conceived of as a tool for simulation language experimentation. That is, as a workbench for the construction of new simulation modeling concepts. Using the set of primitives built into OBSIM and the object oriented programming and rule programming paradigms of the LOOPS language the user can construct new concepts through the following steps:

1. Create a new LOOPS class object with the name of the desired concept.

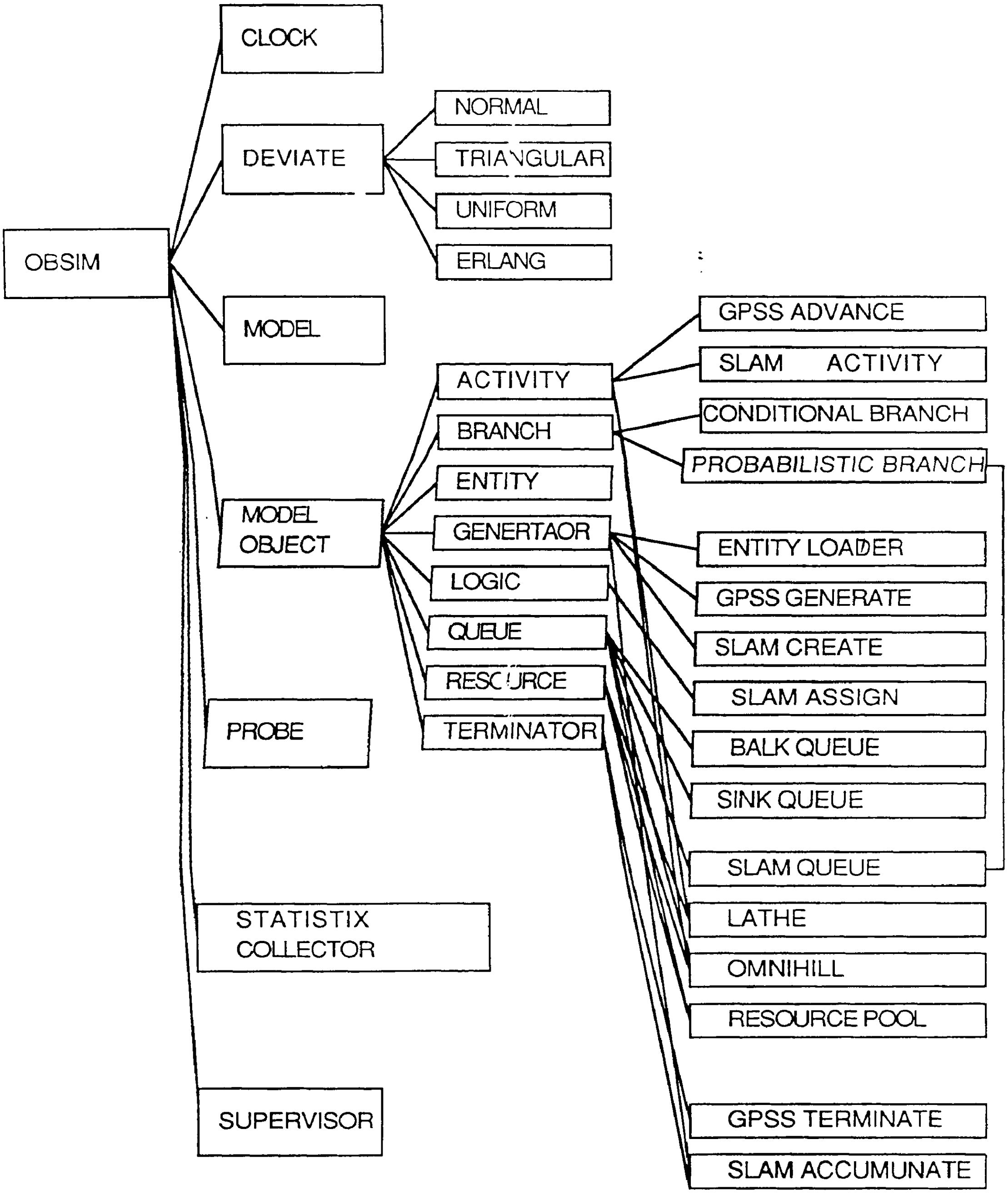


FIGURE 8.37: OBSIM OBJECT HIERARCHY.

2. Assign as "superiors" of that new object, the primitives of the OBSIM language from which the new concept will inherit basic behavior or attribute characteristics.

3. Modify rules of the methods of the mixed in objects to create the specialized behavior of the new concept type.

A simulation model consists of a set of instances of the object classes displayed in Figure 8.37. A simulation run is made by creating an instance of a supervisor object and sending it a message "run." To illustrate the operation of the simulation model, the user creates instances of the "probe" object which automatically creates display gauges such as dials, vertical or horizontal bar graphs, or digital meters which become attached to the particular value of the attribute of the model object which one wishes to display.

8.5.3 Deficiencies / Shortcomings of the Object Paradigm

Object oriented programming was originally introduced as a programming mechanism for modularization of code [Sutherland 1963; Kay 1969; Cannon 1982]. As such, there are definite property and behavior representation shortfalls which arise when attempting to use these programming objects to model real world objects. For instance, the message passing paradigm works quite well in modeling the perceived interaction between a foreman and an operator. But implementation constraints which do not allow for parallel activities (essentially treating the passing of a message as a procedure call) limit the level of behavior representation possible. Similarly, the fact that messages in typical object oriented implementations are not themselves first class objects means that at best they can only be used to define a fixed communication protocol between the agents of the system. More recent extensions to the object oriented paradigm are attempting to solve both of these problems (see [Hewitt 1980] for a discussion of ACTORS). Even with the extensions made in the ACTOR languages there is yet another problem with the object / message paradigm. This problem is due to the fact that many natural constraints of the world do not conveniently fit into the message passing paradigm.

Constraints which reference time and location are particularly notable, for example, the constraint that no two physical objects can occupy the same location at

```
Rule  Set  Activity.Init
Work  Space  Class  :  Activity  ;
Temperory  Variables  :  Requisition  time  ;
Control  Structure  :  DOALL  ;
Args  :  item  ;
(* If activity does not require resources just set up an end of service event *)

If ~resources THEN  entity  ←     item
                     status  ←    busy
                      time  ←   ( ←   . distribution Get sample : parameters)
                     time  ←    time + Tnow
                      (  ←    $ Supervision log self time item  'complete )
                      (  ←    item   update location   self)

IF resources requisition ←  (←  : resources  Request 1)
THEN    entity  ←  item
        status  ←  busy
         time  ←  (←  : : distribution    Get Sample  : parameters)
         time  ←  time + Tnow
        (←  $ Supervision log self time item 'Complete )
        ( ←  item  Update location self )

IF  resources    ~ requisition
THEN
      (  ←    self   Wait For resources )
```

FIGURE 8.38: LOOPS RULE SET FOR OBSIM QUEUE PRIMITIVE INIT METHOD.

the same time as might be desired in the animation of a simulation model or in the representation of the movement of physical parts on a conveyor. Implementation of these types of concepts in the traditional manner requires the entire space of objects to be polled on the change of one objects attributes. While this can certainly be done, it is both computationally expensive and conceptually inelegant. One work around which we have devised is the creation of a separate object for each such "global" property which must be managed. Such a construct allows for enforcement of global constraints to be modeled as communication via shared memory. In actuality, the original object oriented programming implementation in Sketchpad [Sutherland 1963] was totally implemented using the shared memory concepts. Difficulties with the complexity of managing a heterogeneous message traffic precipitated the later movement away from this global memory concept. By structuring the partitioning of such a global memory into heterogeneous chunks, we avoid the management complexity problems.

8.5.4 OBSIM Summary

We have demonstrated the use of object oriented paradigms as a mechanism for constructing a flexible simulation language design system. We have shown that the rule processing capabilities and the provision of both entity tracing and condition/triggering paradigms (which are key requirements on the KAMSS simulation engine) can be accomplished. We have also shown that the object/rule implementation approach allows the integration of a large number of previously considered distinct modeling constructs. Finally, we have shown a mechanism for overcoming a serious problem in the use of object based systems for representing and implementing globally active contraints.

8.6 OBMODLER: An Object Based Model Design Support System

The OBMODLER is an infinite blackboard, menu, and icon driven modeling support system which was prototyped during this research effort. The OBMODLER provides a graphics based alternative to the definition of a system which then can be translated into an OBSIM model automatically. In addition to the standard documentation, storage, retrieval, verification, and translation functions, the design goals of the OBMODLER include the following:

1. Ability to add, change, delete, and move system definition and model components at will on the viewport.

2. Ability to construct the entire model as one interrelated entity on the blackboard and rapidly pan the viewport over the entire set of resulting networks.

3. Ability to have prompting provided for the required parameters of each node type.

4. Ability to select portions of the model to reuse in other models. (ie. ability to select nodes from one model and move them to other models.)

5. Ability to tie the simulation model in its graphics form to the model in its run time state to support the use of the graphical representation in the debugging of a simulation model.

The system definition and model design can be built as the modeler thinks about his problem. This means that no particular order is imposed on the definition of the structures and objects being declared. Nodes are added to the blackboard as the modeler thinks of objects or constructs he needs to include in the system description. The system definition nodes need not be assigned a simulation model concept until the designer desires to incorporate them into the analysis model. Such nodes are merely placed in a boxed or unboxed format on the screen as in Figure 8.39. The modeler has the option to assign a modeling construct to each particular node at creation or simply leave the node on the blackboard as a reminder as shown in Figure 8.40. The system description in the form of a set of linked nodes provides a convenient representation of the system description data which the analyst has acquired.

The modeler can connect the nodes together to form a network which will be interpreted in a manner similar to network simulation systems except that solid physical lines always represent the flow of an object, dashed lines represent the flow of messages, and dotted lines relate merely an association. The modeler can also lasso a group of nodes and assign them a modeling construct to represent a logical grouping of associated description objects with a single model construct.

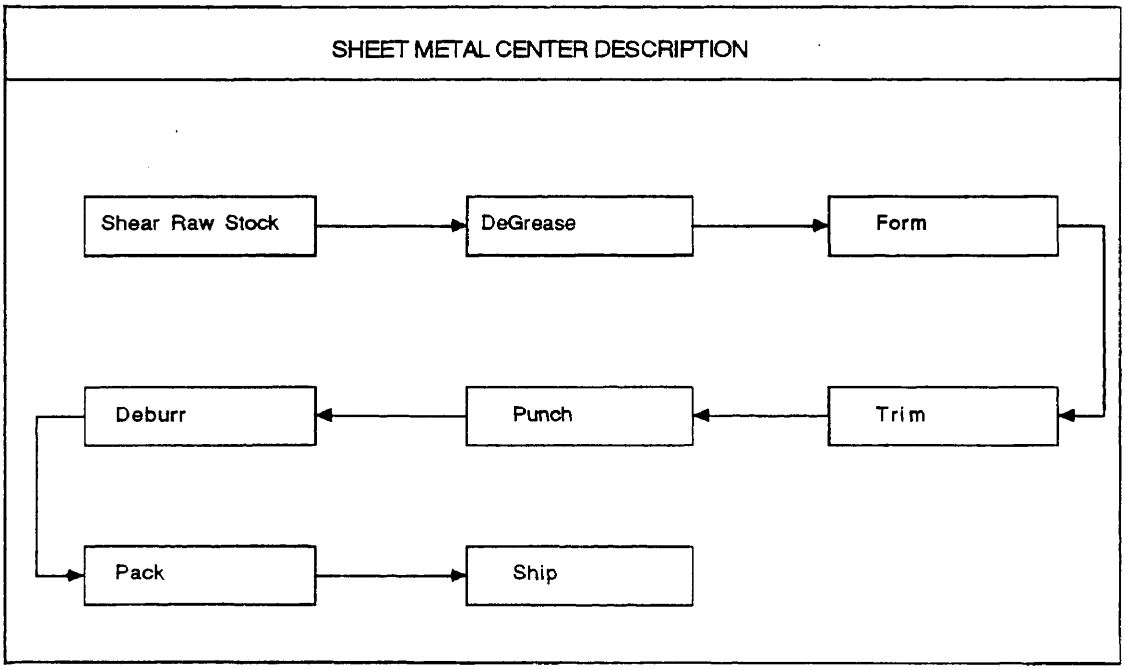


FIGURE 8.39: OBMODLER REPRESENTATION OF A SYSTEM DESCRIPTION.

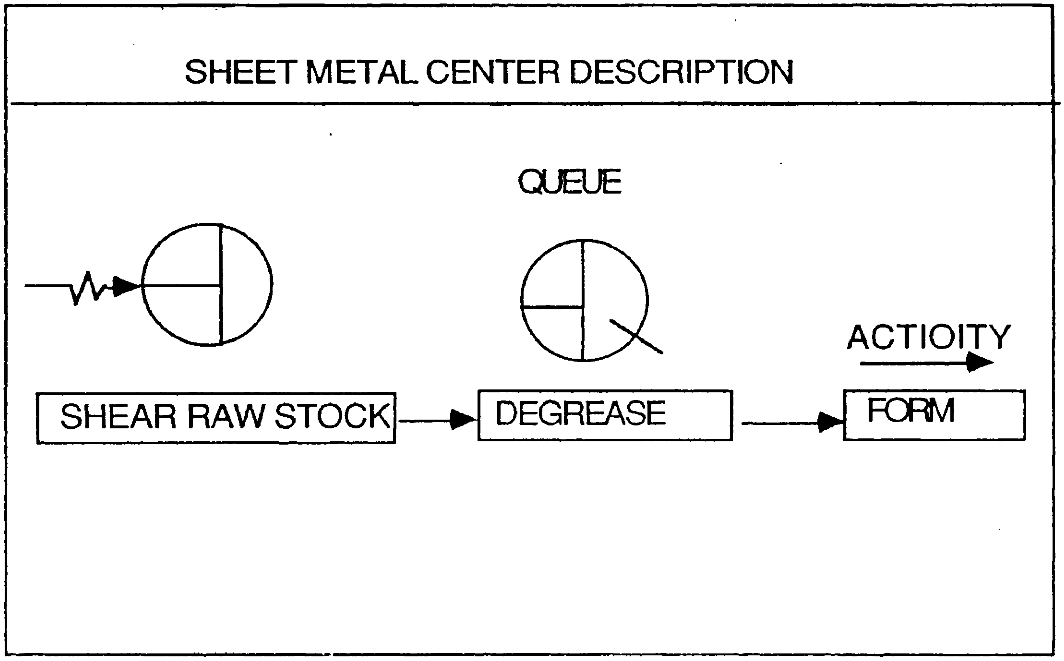


FIGURE 8.40: ASSIGNMENT OF SIMULATION MODEL CONSTRUCTS ON A SYSTEM DESCRIPTION.

The modeler works on a scrollable viewport positioned over a portion of the blackboard. In the model design or editing mode, the modeler can choose from a variety of predefined modeling icons. Some of these icons represent familiar SLAM, GPSS, SIMAN and GEMS symbol types. Others represent predefined physical object types, such as a 5-axis Sundstrand Omnimill or a Local Area Communication Network. Still others represent complete canned models such as a grinding cell with its own material handling, inprocess storage, and control logic. One of the top level features provided by OBMODLER is the capability for the modeler to add to or remove from this set of primitive icon symbols. For example, if the modeler finds it convenient to add the SLAM GOON node as a primitive, he can construct the icon which represents that node, build the translation of that node into the OBSIM language, and add it to the list of icons. The icons are then displayed when the initial (OPENSESAME) command is given. The modeler can also open up displays of previously constructed system descriptions or models and "lasso" subsections of those models and pull the selected parts into his current work area (see Figure 8.41).

The resulting model is set up in such a way that it can also be used to monitor the results of an ongoing simulation in several interesting ways. Building upon the active value implementation of the data-driven programming concepts of LOOPS, any attribute value of a simulation modeling object can be defined as an active value. The active value can trigger a function when it is accessed or changed or both. Even the blackboard representation of that node can be affected. Thus, for example, activities can be made to invert to white lettering on black background when the activity is busy. Also, a set of modeling objects called probes are provided which allow the modeler to indicate that a value or a particular set of values is to be continuously monitored and displayed during the execution of a simulation. The association of a probe with a node attribute would cause a gauge to be displayed which would show the value of the attribute during the simulation. The gauge itself is an object which the modeler can tailor to his specific requirements. While there are many built-in gauges (linear vertical and horizontal gauges, round dial gauges, simple LED gauges and pointer gauges), it is a simple matter to mix these types together or to create a new class of gauges.

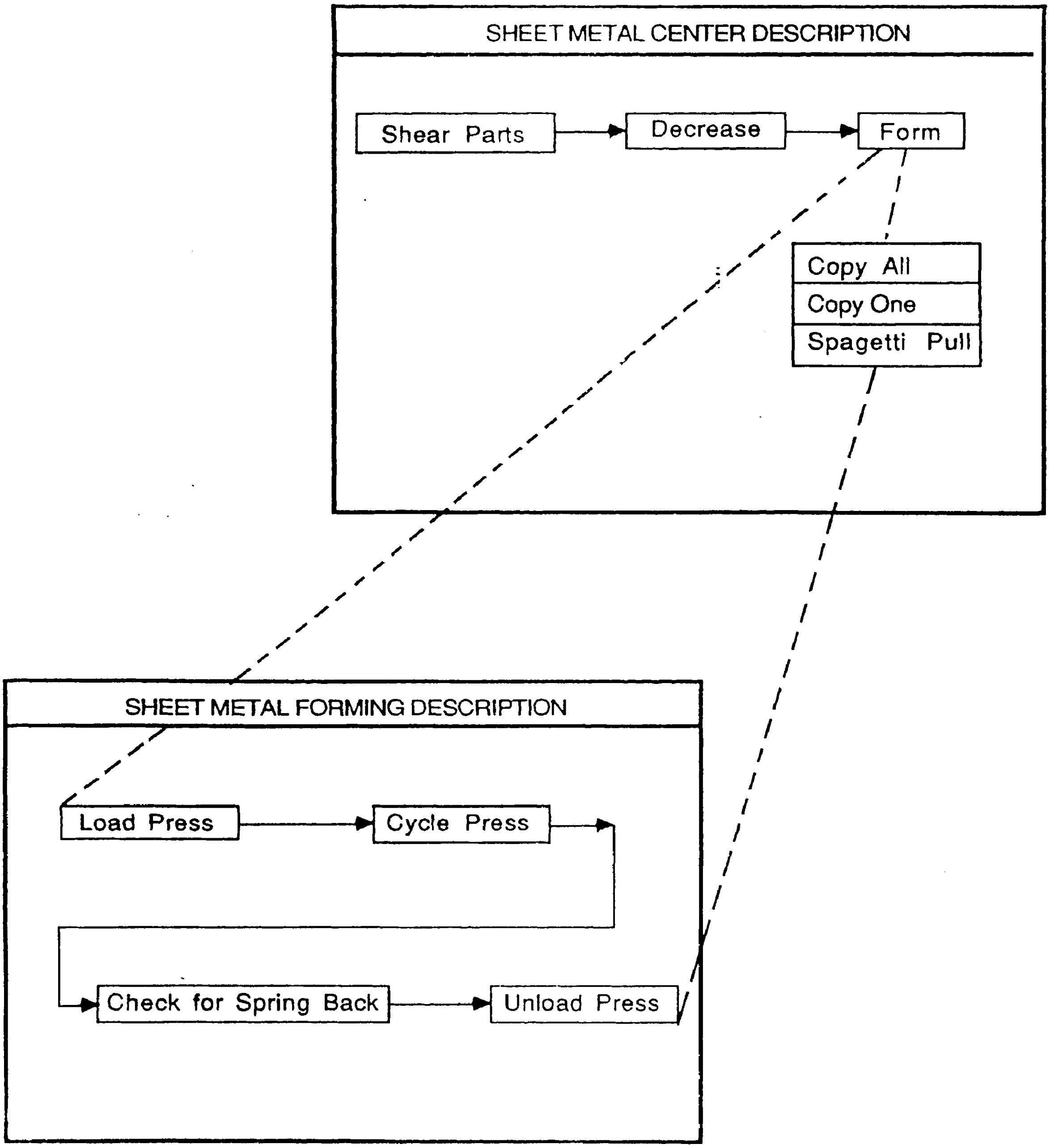


FIGURE 8.41: REUSING PREVIOUSLY DEFINED SYSTEM DESCRIPTIONS OR MODELS.

8.6.1 OBMODLER Summary

We have demonstrated that the separation of system description and model design in an integrated support environment can be accomplished. We have also shown that the object/rule implementation approach allows the integration of a large number of previously considered, distinct modeling constructs. Finally, we have implemented in the OBMODLER a support tool which integrates these concepts with an infinite electronic blackboard for definition and display.

9. CONCLUSION

In this dissertation, we presented the hypothesis that the primary view of simulation by non-simulation analysts is one of *reasoning about the structure of a system* or *reasoning about implications* of a proposed change to a system. This hypothesis can be rationalized by examination of the cognitive processing activities required to understand, plan, recognize problems with, and evolve a manufacturing system. With this perspective in mind and the tools of knowledge based systems, we can consider the development of a new generation of system modeling and analysis support environments. The environments envisioned would essentially encompass the knowledge and skills of a human systems analyst as well as the analysis tools of existing systems. However the problems associated with realizing such environments are extensive. What we have presented in this work is a description of the cognitive tasks which must be supported, a conceptual approach to the development of such a capability, an architecture for an operational system of this complexity, an outline of the basic semantic and reasoning concepts which underly this architecture, and some of the tools needed to realize the eventual system.

9.1 Summary of Contributions

The results presented in each section of this dissertation are:

Section 1 was meant to establish the rational for the view that current modeling and simulation support system concepts are inadequate for the task at hand. In this section we also pointed out that for the techniques of knowledge based systems to be profitably applied to this problem area, we must first attempt to apply the methods of AI to the problem.

In Section 2 we have attempted to characterize the kinds of cognitive activities which a human goes through in order to understand his environment, recognize its deficiencies, and construct and use models as mechanisms for dealing with those problems. We showed how our observations and experiences in these activities could be related to prevailing AI models of human cognitive activities from language understanding to planning and common sense reasoning about physical systems. This characterization has merit, but from an engineering standpoint, we can use it as a definition of requirements for a next generation modeling and support package as discussed in Section 3.

In Section 3 we developed the requirements, philosophy of operation, and architecture for a knowledge acquisition and modeling support environment (KAMSS) which could support a broad range of the cognitive activities identified in Section 2.

In Section 4 we examined the issues of natural language processing from three points of view: understanding (NLU), generation (NLG), and discourse management (DM). While the general problem of natural language understanding is very difficult, we presented workable approaches to the subset of that processing which would be sufficient for the requirements of a system like KAMSS. More importantly we developed a method, though originally conceived for the analysis required to build natual language interfaces, that can be used for the definition and design of the knowledge bases within the KAMSS system.

In Section 5 we present the general issues associated with the representation and manipulations of the "meaning" of symbols. We proposed a basis for a theory of semantics which can usefully be employed to support the concept discovery and reasoning required of the KAMSS. This theory is a natural extension of the "Situation Semantics" work [Barwise and Perry 83].

In Section 6 we present a reasoning method which combines previous concepts in belief revision [Harmon 86] with syntactic reasoning methods [Hass 86] to provide a new approach which views reasoning as a process of the construction of information chains. We showed that such an approach can accommodate the types of reasoning required in KAMSS including both traditional deductive and inductive methods as special cases.

In Section 7 we came back to the issues of "what constitutes a model" and the process of modeling itself. The purpose was to illustrate that armed with the theories of semantics and reasoning from Sections 5 and 6, the mathematical notions of modeling and simulation could be shown to be special cases of a more general process of reasoning about the consequences of situations in the world around us. This section also illustrates how the techniques proposed can provide a usable framework for model design automation without resorting to pre-canned, parameterized models.

In Section 8 we describe the results of prototyping activities which were undertaken to support critical areas or claims within this dissertation. These prototypes were meant to illustrate the engineering soundness of the concepts presented and to serve as proof of engineering prototypes for the construction of the actual KAMSS system.

9.2 Areas for Further Research

In retrospect, the work presented in this dissertation has raised as many questions as it has answered. Every issue resolved would give rise to yet more untouched issues. In the terms of Section 5, the bindings are deeply nested. The following are a few of the areas which need more detailed investigation before KAMSS can be considered a reality:

1. There is a need for detailed IDEF0 models for each major mode of operation of the KAMSS. The IDEF0 modeling of the cognitive activities identified in Section 2 would be the basis for starting this process.

2. There is a need for more IDEF1/ES models of the manufacturing system description domain from the various viewpoints characterized in Section 3. These models are required to complete the design of the manufacturing view of the KAMSS knowledge base.

3. There is a need for more work in the conceptualization of the dialog manager, particularly in the area of methods for interaction focusing. This area appears to be lacking in published works of any significance. Even the natural language processing literature has overlooked this important area.

4. The cognitive activities involved in the interpretation of model results presented in Section 5 are sketchy at best. This area is much more complex than originally anticipated and could use a great deal more research.

5. More work on the abstraction mechanisms in the design of models from system descriptions is needed before the MDS can be extended into the model generation domain. This work represents approximately six man-months of detailed knowledge engineering with several modeling experts.

6. Relative to the issues of an adequate theory of semantics there is considerable additional work required before the basis presented can be implemented in a generalized support mechanism. These enhancements include:

   6.1. Develop a formal ontology for product descriptions that include form/ feature representations.

   6.2. Enhance the notion of attitudes to account for three levels of knowledge possession (action enabling possession, buzzword possession, display possession).

   6.3. Provide a theory of reliable acquisition of knowledge as a precondition to behavior enabling possession of knowledge.

   6.4. Resolve the issues of knowledge acquisition with the information conveyed by an utterance versus the information displayed by that utterance.

   6.5. Account for sentence negation.

   6.6. Account for customer beliefs which change over time.

   6.7. Elaborate the concept of "stakes" to take into account:

      6.7.1. Partial orderings versus linear time.

      6.7.2. Law of strict causality.

      6.7.3. Absolute time relations (time sequence).

      6.7.4. Optionality of global time.

   6.8. Characterize the semantics of the man-machine interface.

7. Relative to the issues of an adequate theory of reasoning, there is considerable additional work required before the basis presented can be implemented in a generalized support mechanism. These enhancements include:

   7.1. Formalize the adaptive reasoning approach which can be shown to be able to generate the classical logic systems in the context of certain observations. Thus, rather than attempting to show that our reasoning method is "complete" or "sound," we would show that our reasoning method could result in the discovery of formal logic methods.

7.2. Develop an axiomization of the discovery and method components of the AR concept presented in Section 6.

7.3. Characterize the process of reasoning about another agents beliefs.

7.4. Extend the notion of contraint reasoning to include reasoning about cause and affect as the basis for the use of mechanistic reasoning in model design.

7.5. Define analogical reasoning and its role in model reuse.

7.6. Describe constraints as the means by which events are propagated.

8. The treatment of natural language understanding and generation only began to scratch the surface of the methods required to provide a robust version of a KAMSS. Much of the available research and tools are derivatives of syntactic theories. Linguistics considerations for semantics and the pragmatics of language are only just beginning to emerge. There is a need to evaluate the syntactic reasoning mechanisms and the semantics presented in Sections 5 and 6 with the concepts presented in Section 4 in order to develop a sound theoretical basis for a set of usable tools to implement NLP systems.

9. There is a need to extend the concepts developed in this dissertation to other modeling and analysis methods in operations research. An ultimate goal of KAMSS is to support the selection of the "best" modeling technique for the problem at hand, and not merely brute force the simulation of every problem situation.

10. The system description knowledge base which is constructed for KAMSS could easily support many of the system engineering decision activities in a manufacturing system or software development process. It would be very interesting to consider the KAMSS architecture as the basis for an integrated system development support environment.

9.3 Implementation Considerations

Before a full scale implementation of the KAMSS is possible, there are at least three other prototype activities necessary to organize the full scale development. These are:

1. A more robust Qualitative Reasoner from System Descriptions (QUARS),
2. Model Generation from System Descriptions (MODGEN) for both qualitative and quantitative models,
3. Situation Based System Description Knowledge Database (SDKD).

QUARS must be extended to demonstrate the feasibility of applying the syntactic information chain reasoning concepts presented in Section 6 to system descriptions whose representation was consistent with the SRM concepts presented in Section 5. The pattern matching features, context mechanisms, and schema inheritance capabilities of ART on the Symbolics 3640 would make this the logical tool for the prototype implementation. However, as discussed in Section 3 of this report, there are several limitations on this implementation mechanism which would probably prohibit a scaled up version of this prototype. The availability of the Symbolics Joshua toolkit combined with the current SDCE prototype would provide the basic structure for further research on this problem.

MODGEN must be designed to demonstrate the feasibility of directly generating simulation models from system descriptions provided in the SDCE format. A combination of the Symbolics products Joshua and Statice could be used as the development environment combined with a rudimentory discourse manager, discourse generator, and concept manager. These tools would be written in Common Lisp and Flavors on the Symbolics 3640 for easy integration with the Symbolics tools.

The SDKD was actually prototyped as a part of the SDCE where the primary issues were representation completeness and access efficiencies. What is needed is a stand alone implementation concerned with the issues of efficiencies for large description databases and distribution of such data bases. Based on the structure presented for descriptions as named collections of noticed characteristics (bags of facts), an approach which combined predications from Joshua for the logic based fact types with partially populated Rete nets for the syntactic based fact types with instantiated instance variables for the procedural fact types is probably the direction which must be taken. As Statice supports each of these structures it would probably serve as a logical platform for such an experiment.

REFERENCES

Adelsberger, H. H., and Neumann G. 1985. Goal Oriented Simulation Using Prolog. *Proceedings of the 1985 SCS Conference on Modeling and Simulation on Micro-Computers,* SCS, San Diego, CA, (Jan.).

Adelsberger, H. H., Pooch, U., Shannon, R., and Williams, G. 1985. Rule Based Object Oriented Simulation Systems. *Proceedings AI, Graphics, and Simulation Conference,* SCS, San Diego, CA, (Jan.).

Aiello, N., Bock, C., Nii, H., and White W. 1981. AGÈ Reference Manual AGE–1. *Heuristic Programming Project.* Computer Science Department, Stanford University, Stanford, CA.

Alford, M., Smith, T., and Smith, D. 1979. Formal Decomposition Applied to Axiomatic Requirements Engineering. Final Report, Project # 34674-6921-009. TRW Defense and Space Systems Group, Huntsville, AL.

Allen, B. P., and Wright, J. M. 1983. Integrating Logic Programs and Schemata. *Proceedings of the Eighth International Joint Conference on Artificial Intelligence.* Morgan Kaufmann Publishers Inc. San Mateo, CA, pp. 340 - 342.

Allen, J. 1987 *Natural Language Understanding.* The Benjamin/Cummings Publishing Company, Inc., Menlo Park, CA.

Appelt, D. E. 1980. A Planner for Reasoning about Knowledge and Action. *Proceedings of the First Annual National Conference on Artificial Intelligence*, Morgan Kaufmann Publishers Inc. San Mateo, CA, pp. 131 - 134.

Balzar, R. M., and Goldman N. M. 1982. Operational Specification as the Basis for Rapid Prototyping. *Proceedings Second Software Engineering Symposium.* ACM SIGSOFT, (April).

Barber, G. R. 1982. Office Semantics. Ph.D. dissertation, Dept. of Electrical Engineering and Computer Science, Massachesetts Institute of Technology, Cambridge MA.

Barwise, J. 1984. The Situation in Logic – I. Report No. CSLI-84-2, The Center for the Study of Language and Information, Stanford University, Stanford, CA.

Barwise, J. 1985a. The Situation in Logic – II: Conditionals and Conditional Information. Report No. CSLI-84-2, The Center for the Study of Language and Information, Stanford University, Stanford, CA.

Barwise, J. 1985b. The Situation in Logic – III: Situations, Sets and the Axiom of Foundation. Report No. CSLI-84-2, The Center for the Study of Language and Information, Stanford University, Stanford, CA.

Barwise, J., and Perry, J. 1983. *Situations and Attitudes.* MIT Press, Cambridge, MA.

Barwise, J., and Perry, J. 1985. Shifting Situations and Shaken Attitudes. Report No. CLSI-85-13, The Center for the Study of Language and Information, Stanford University, Stanford, CA.

Baskaran, V., and Reddy, Y. V. 1984. An Introspective Environment for Knowledge Based Simulation. *Proceedings of the 1984 Winter Simulation Conference*, SCS, Dallas, TX, (Nov.).

Benzon, W. 1987. Reactions to Darden. *AI Magazine* Vol. 8, No. 4, Letters, Winter.

Bobrow, D. G., ed. 1985. *Qualitative Reasoning About Physical Systems.* MIT Press, Cambridge, MA.

Bobrow, D. G., and Collins, A. 1975. In *Representation and Understanding: Studies in Cognitive Science.* Bobrow and Collins, Eds. Academic Press, Inc., Orlando, FL.

Bobrow, D. G., and Stefik, M. 1983. *The LOOPS Manual.* Intelligent Systems Laboratory, Xerox Corporation, Palo Alto, CA.

Bobrow, D. G., and Winograd, T. 1977. An Overview of KRL, A Knowledge Representation Language. *Cognitive Science*, Vol. 1, No. 1.

Brachman, R. J. 1983. What IS-A Is and Isn't: An Analysis of Taxonomic Links in Semantic Networks. *Computer.* (Oct.) pp. 30-36.

Brachman, R. J. 1985. 'I Lied About the Trees' Or, Defaults and Definitions in Knowledge Representation. *The AI Magazine.* (Fall) pp. 80 – 93.

Brachman, R. J., Fikes, R. E., and Levesque, H. J. 1983. KRYPTON: A Functional Approach to Knowledge Representation. *IEEE Computer.* (Oct.) pp. 67 – 73.

Bunt, H. 1985. The Formal Representation of (Quasi-) Continuous Concepts. In *Formal Theories of the Commonsense World,* J. Hobbs and R. Moore Ed. Ablex Publishing Corporation, Norwood, NJ.

Campbell, J. 1984. *Implementations of Prolog.* J. A. Cambell ed., Ellis Horwood ltd., John Wiley and Sons, New York, NY.

Cannon, H. 1982. Flavors, A non-hierarchical approach to object-oriented programming. Unpublished report, Copyright (c) 1982 by Howard I. Cannon, Symbolics, Inc. Cambridge MA.

Carnegie Group. 1986. The Language Craft Reference Manual Version 3.1. Carnegie Group Inc., Pittsburgh, PA.

Carnegie Group. 1987. The Knowledge Craft Reference Manual Version 3.1 Vols. 1, 2, & 3. Carnegie Group Inc., Pittsburgh, PA.

Cleary, J., Goh, K., and Unger, B. 1985. Discrete Event Simulation in Prolog. *Proceedings AI, Graphics, and Simulation Conference.* SCS, San Diego, CA, (Jan.).

CLIPS Reference Manual, Version 3.0. 1986. Mission Planning and Analysis Division, Artificial Intelligence Section, Johnson Space Center, NASA.

Clocksin, W., and Mellish, C. 1981. *Programming in Prolog.* Springer-Verlag, New York, NY.

Cohen, P., and Feigenbaum, E. A. 1981. *The Handbook of Artificial Intelligence.* Vol. III. William Kaufman, Inc., Los Altos, CA.

Corynen, G. C. 1975. A Mathematical Theory of Modeling and Simulation. Phd. Disertation, Department of Engineering and System Science, University of Michigan, Ann Arbor, MI.

Davey, A. 1978. *Discourse Production: A computer model of some aspects of a speaker.* Edinburgh University Press, Edinburgh, United Kingdom.

Davis, R. 1984. Reasoning from First Principles in Electronic Troubleshooting. *Developments in Expert Systems.* Academic Press, pp. 1 – 21.

Davis, R. and Brown J. 1984. Qualitative Physics Based On Confluences. *Artificial Intelligence* 24. pp. 7 – 83.

Deshler, M. 1981. The IDSS 1.4 User Reference Manual. Contract # F33615-78-C-5231, United States Air Force, AFWAL/MLTC. WPAFB, OH.

Doyle, J. 1979. A Truth Maintenance System. *Artificial Intelligence* 12. pp. 231-272.

Dyer, M. 1983. *In-Depth Understanding, A Computer Model of Integrated Processing for Narrative Comprehension.* MIT Press, Cambridge, MA.

Elmaghraby, A. S., and Jangannathan V. 1985. An Expert System for Simulationists. *Proceedings AI, Graphics, and Simulation Conference.* SCS, San Diego, CA, (Jan.).

Fahlman, S. E. 1985. *NETL A System for Representing and Using Real-World Knowledge.* The MIT Press, Cambridge, MA.

Fikes, R., and Kehler, T. 1985. The Role of Frame-Based Representation in Reasoning. *Communications of the ACM.* Vol. 28, No. 9, pp. 904 – 920.

Fillmore, C. J. 1968. The Case for Case. *Universals in Linguistic Theory.* Bach and Harms, Eds., Holt, Rinehart, and Winston, Inc., New York, NY.

Fodor, J. D. 1977. *Semantics: Theories of meaning in generative grammar.* Harvester Press, Hassocks, Sussex.

Forbus, K. D. 1984. Qualitative Process Theory. *M.I.T. AI Laboratory Technical Report No. 789* M.I.T., Cambridge, MA.

Frege, G. 1960. On sense and reference. In *Translations from the Philosophical Writings of Gottlob Frege.* P. Gretch and M. Black Eds. Oxford: Basil Blackwell.

Friedman, J. 1966. Directed random generation of sentences. *Communications of the ACM*, 7:2, pp. 40 – 46.

Friel, P. G., 1987. Automotive Cooling System Designer. KBS Technical Report. Chrysler Motors Corporation, Detroit MI.

Friel, P. G., and Mayer, R. J. 1985. Life Cycle Methods, Tools, and Environments for Support of the Expert System Development Process.*Proceedings First Annual Workshop on Robotics and Expert Systems.* NASA & Instrument Society of America, Clearlake, TX.

Futo, I. 1984. System Simulation and Co-Operative Problem Solving on a Prolog Basis. *Implementations of Prolog.* J. A. Cambell ed., Ellis Horwood Ltd., John Wiley and Sons, New York, NY.

Futo, I. 1985. Combined Discrete/Continuous Modeling and Problem Solving. *Proceedings AI, Graphics, and Simulation Conference.* SCS, San Diego, CA, (Jan.).

Gaines, B. R., and Shaw, M. L. G. 1985. Expert Systems and Simulation. *Proceedings AI, Graphics, and Simulation Conference.* SCS, San Diego, CA, (Jan.).

Genesereth, M. R., and Ginsberg, M. L. 1985. Logic Programming. *Communications of the ACM.* Vol. 28, No. 9, pp. 933–941.

Grishman, R., and Kittredge, R., Eds. 1986. *Analyzing Language in Restricted Domains: Sublanguage Description and Processing.* Lawrence Erlbaum, Hillsdale, NJ.

Hass, A. R. 1986. A Syntactic Theory of Belief and Action. *Artificial Intelligence*, Volume 28, No. 3, pp. 245 - 292.

Harman, G. 1986. *Change in View, Principles of Reasoning.* The MIT Press, Cambridge, MA.

Hayes, J. 1979. The Naive Physics Manifesto. In *Expert Systems in the Micro-Electronic Age,* D. Michie Ed. Edinburgh University Press, Edinburgh, Scotland.

Hayes–Roth, B., and Hayes–Roth, F. 1978. Cognitive Processes in Planning. ONR Report # R-2366-ONR.

Heidorn, G.E. 1972. *Natural Language Inputs to a Simulation Programming System.* Naval Postgraduate School, Monterrey, CA.

Hewitt, C. 1980. The Apiary Network Architecture for Knowledgeable Systems. In *Conference Record of the 1980 Lisp Conference.* Stanford University, Stanford CA.

Hobbs, J. 1985. Introduction. In *Formal Theories of the Commonsense World*, J. Hobbs and R. Moore Eds. Ablex Publishing Corporation, Norwood, NJ.

Hobbs, J. 1986. On the Coherence and Structure of Discourse. In *The Structure of Discourse*, L. Polanyi Ed. Ablex Publishing Corporation, Norwood, NJ.

Hobbs, J., Blenko, T., Croft, B., Hager, G., Kautz, H., Kube, P., and Shoham, Y. 1985. Commonsense Summer: Final Report. Report No. CLSI-85-35, The Center for the Study of Language and Information, Stanford University, Stanford, CA.

Hobbs, J., Croft, W., Davies, T., Edwards, D., and Laws, K. 1987. The TACITUS Commonsense Knowledge Base (Draft). Artificial Intelligence Center SRI International, Menlo Park, CA.

IDEF0. 1980. IDEF0 Short Course, KBS Laboratory, Department of Industrial Engineering, Texas A&M University, College Station TX.

IDS. 1987. Integrated Design Support Prospectus, IDS Program Office, United States Air Force, AFWAL / FIBAB, Wright-Patterson Air Force Base, OH, 45433, August.

IISS. 1983. ICAM Integrated Information Support System Test Bed System Design Specification. SDS620140000, AFWAL /MLTC, WPAFB, OH.

Inference Corporation. 1985. *ART Reference Manual.* Inference Corporation, Los Angeles, CA, (April).

IntelliCorp. 1985a. *KEE Software Development System User's Manual.* IntelliCorp, Menlo Park, CA.

IntelliCorp. 1985b. *The SIMKIT System, Knowledge Based Simulation in KEE.* IntelliCorp, Menlo Park, CA.

International Standards Organization. 1981. Concepts and Terminology for the Conceptual Schema. Preliminary Report. ISO TC 97/SC5/WG5, edited by J. J. van Griethushsen, (Feb.).

Israel, D. J. 1983. The Role of Logic in Knowledge Representation. *IEEE Computer.* (Oct.) pp. 37–41.

Kaisler, S.H. 1986. INTERLISP. John Wiley and Sons, NY.

Katz, J.J. 1980. *Propositional Structure and Illocutionary Force: A Study of the Contribution of Sentence Meaning of Speech Acts.* Harvard University Press, Cambridge, MA.

Kay, A. 1969. The Reactive Engine. Ph.D. Thesis, University of Utah, Salt Lake City, UT.

Kittredge, R., and Lehrberger, J. 1982. *Sublanguage: Studies of Language in Restricted Semantic Domains.* Walter de Gruyter, Berlin.

Kiviat, P. J., Villanueva, R., and Markowitz, H. M. 1973. Simscript II.5 Programming Language. C.A.C.I., Los Angeles, CA.

Ko, H., and Wheeler, J. 1983. A Computer Simulation Methodology by LISP and SANS. *Proceedings of the Summer Computer Simulation Conference.* , SCS, San Diego, CA, (July).

Kowalski, R. 1979. *Logic for Problem Solving.* North-Holland, New York, NY.

Kowalski, R. 1981a. Logic for Data Description. Internal Research Paper. Imperial College, London.

Kowalski, R. 1981b. Amalgamating Language and Meta-Language in Logic Programming. Internal R&D Report. School of Computer and Information Science, Syracuse University, Syracuse, NY.

Krishnamurthi, M., Mayer, R. J., and Friel, P. G. 1986a. Integrating Expert Systems into Engineering Environments. *Proceedings of the 1986 IEEE AI Workstation & Systems Technology Conference.* (Atlantic City, NJ.). IEEE, New York, (Jan.).

Krishnamurthi, M., Mayer, R. J., and Friel, P. G. 1986b. Machine Fault Diagnosis: Combining Deep and Shallow Modeling Approaches for Practical Extensible Applications. *Proceedings of the 1986 SME Ultratech Conference.* (Long Beach, CA.). SME, Dearborn, MI, (Aug.).

Krishnamurthi, M., Mayer, R. J., and Friel, P. G. 1986c. Robot Diagnosis Using Deep and Shallow Modeling Approaches. *Proceedings of ROBEXS '86.* Instruments Society of America, Houston, TX.

Kuipers, B. 1984. Commonsense Reasoning About Causality: Deriving Behavior from Structure. *Artificial Intelligence* 24. pp. 169 – 123.

Kuipers, B. and Patil, R. 1987. Qualitative Simulation and Causal Models. *National Conference on Artificial Intelligence Tutorial No: HA 4.* (Seattle, WA.). Morgan Kaufmann Publishers Inc. San Mateo, CA, pp. 7 – 9.

Lenat, D., Prakash, M., and Shepherd, M. 1986. CYC: Using Common Sense Knowledge to Overcome Brittleness and Knowledge Acquisition Bottlenecks. AI Magazine, Vol. 24, No. 4, pp 65 – 84

Liu, D. 1987. Intelligent Manufacturing Planning Systems. In *Smart Manufacturing with Artificial Intelligence.* Computer and Automated Systems Association of SME, Dearborn, MI.

Lloyd, J. 1984. *Foundations of Logic Programming.* Springer-Verlag, New York, NY.

Marcus, M. 1980. *A Theory of Syntactic Recognition for Natural Language.* MIT Press, Cambridge, MA.

Markowitz, H., Malhotra, A., and Pazel, D. 1978. EAS-E: An Executable Application Design Language. IBM Research Report #RC7349 (#31614). (Oct.). Department of Computer Science, IBM Thomas J. Watson Research Center, Yorktown Heights, NY.

Mayer, P. S. D. 1988. A Computational Approach for Processing Locative and Temporal Information in Clinical Medical Records. Phd Dissertation, Department of Computer Science, Texas A&M University, College Station, TX.

Mayer, P. S. D., Bailey, G., Mayer, R. J., Hillis, A., and Dvoracek, J. 1987. Locative Inferences in Medical Texts. *Proceedings of the 1987 Hawaii International Conference on System Sciences.* (Kailua-Kona, Hawaii.). Western Periodicals Company, North Hollywood, CA. (Jan.).

Mayer, R. J. 1983. Simulation Model Generation from System Specifications. KBS Technical Report, Knowledge Based Systems Laboratory, Department of Industrial Engineering, Texas A&M University, College Station, TX.

Mayer, R. J. 1985. IDEF1 Short Course, KBS Technical Report, Knowledge Based Systems Laboratory, Department of Industrial Engineering, Texas A&M University, College Station TX.

Mayer, R. J. 1986. Design Concepts of the OBSIM and OBMODELER systems. KBS Technical Report, Knowledge Based Systems Laboratory, Department of Industrial Engineering,Texas A&M University, College Station, TX.

Mayer, R. J., et al. 1988. Integrated Information System Evolution Methodologies and Environments. KBS Technical Report. U.S. Air Force AFWAL/MS, WPAFB, OH.

Mayer, R. J., Friel, P. G., Krishnamurthi, M., and Underbrink, A., 1986. A Characterization of Expert System Development Tools for Manufacturing Applications. KBS Technical Report. U.S. Air Force AFWAL/MS, WPAFB, OH.

Mayer, R. J., Friel, P. G., Krishnamurthi, M., Underbrink, A., and Wells, M., 1987. Artificial Intelligence Applications in Automotive Production. KBS Technical Report. Chrysler Motors Corporation, Detroit MI.

Mayer, R. J., and Young, R. 1984. Simulation Model Generation from System Specifications. *Proceedings of the Winter Simulation Conference*, SCS, Dallas, TX, (Nov.).

McArthur, D. 1981. An Object-Oriented Language for Constructing Simulations. *Proceedings of the Seventh International Conference on Artificial Intelligence.* Morgan Kaufmann Publishers Inc. San Mateo, CA, pp. 809 - 814.

McDermott, D. 1985. Reasoning about Plans In *Formal Theories of the Commonsense World,* J. Hobbs and R. Moore Ed. Ablex Publishing Corporation, Norwood, NJ.

McDonald, D. 1982. Natural Language Generation as a Computational Problem. In *Computational Models of Discourse,* M. Brady Ed. MIT Press, Cambridge MA.

McRoberts, M., Fox M., and Husain, N. 1985. Generating Model Abstraction Scenarios in KBS. *Proceedings AI, Graphics, and Simulation Conference.* SCS, San Diego, CA, (Jan.).

Mellish, C. 1985. *Computer Interpretation of Natural Language Descriptions.* Ellis Horwood Limited, Halsted Press, Chichester, UK.

Mesarovic, M. D., Macko, D., and Takahara, Y. 1970. *Theory of Hierarchical, Multilevel, Systems.* Academic Press, New York, NY.

Mesarovic, M. D., and Takahara, Y. 1975. *General Systems Theory Mathematical Foundations.* Academic Press, New York, NY.

Minsky, M. 1974. A Framework for Representing Knowledge. MIT AI Memo 306, Cambridge, MA.

Moore, R. 1980. Reasoning about knowledge and action, Technical Report 191, SRI International, Menlo Park, CA.

Morrison, K. 1986. *Requirements for a Manufacturing System Description Capture Environment (SDCE).* Chrysler Motors Outer Drive Manufacturing Technology Center, Detroit MI.

Nance, R. E. 1981. The Time and State Relationships in Simulation. *Communications of the ACM.* Volume 24, No. 4, April.

Nelson, S. S. 1977. Control Issues in the Development of a Conversational Simulation Language. Ph.D. dissertation, Dept. of Computer Science, University of Pittsburgh, Pittsburgh, PA.

Nijssen, G. M. 1982. An Architecture for Knowledge Representation. Internal Research Report. Control Data Corporation, Europe, Brussels, Belgium.

Nilsson, N. J. 1980. *Principles of Artificial Intelligence.* Tioga Publishing Company, Palo Alto, CA.

O'Keefe, R. 1986. Simulation and Expert Systems – A Taxonomy and Some Examples. *Simulation* 46:1.

O'Shea, T., Eisenstadt M. 1984. *Artificial Intelligence,* Harper & Row, New York, NY.

Overstreet, M., and Nance, R. 1985. A Specification Language to Assist in Analysis of Discrete Simulation Models. *Communications of the ACM,* Vol. 28, No. 2.

Pegden, C. D. 1982. *Introduction to SIMAN.* Systems Modeling Corporation, State College, PA.

Phillips, D. T. 1979. GEMS Model of a Tool Joint Manufacturing System. Report No. GEMS-11-79, NSF/ASRA Grant No. APR 76-22610.

Pritsker, A. A. B. 1977. *Modeling and Analysis Using Q-GERT Networks.* John Wiley and Sons, New York, NY.

Pritsker and Associates Inc. 1983. The IDSS Prototype (2.0) Users Reference Manual. Pritsker and Associates, West Lafayette, IN.

Pritsker and Associates Inc. 1984. The IDSS Build 1 Final Report. Pritsker and Associates, West Lafayette, IN.

Pritsker, A. A. B., and Pegden, C. D. 1979. *Introduction to Simulation and Slam.* John Wiley and Sons, New York, NY.

Pritsker, A. A. B., and Young, R. E. 1975. *Simulation With GASP/PLI.* John Wiley and Sons, New York, NY.

Ramey, T. L. 1981. ELKA Information Modeling. Internal Research Report. Hughes Aircraft Co., Fullerton, CA.

Ramey, T. L. 1983. GuideBook to Systems Development. Internal Research Report. Hughes AirCraft Co., El Segundo, CA.

Reddy, Y. V., Fox, M. S., and Husain, N. 1985. Automating the Analysis of Simulations in KBS. *Proceedings AI, Graphics, and Simulation Conference.* SCS, San Diego, CA, (Jan.).

Reilly, K., Jones, W., and Dey, P. 1985. The Simulation Enviroment Concept, Artificial Intelligence Perspectives. *Proceedings AI, Graphics, and Simulation Conference.* SCS, San Diego, CA, (Jan.).

Ritchie, G. D. 1984. AM: A Case Study in AI Methodology *Artificial Intelligence* 23. pp. 249 – 268.

Sacerdoti, E. 1977. *A Structure for Plans and Behavior.* Elsevier, New York, NY.

Sager, N. 1981. *Natural Language Information Processing.* Addison-Wesley, Reading, MA. 1981.

Schank, R. 1975. *Conceptual Information Processing.* North-Holland, Amsterdam.

Schank, R. 1982. *Dynamic Memory.* Cambridge University Press, New York, NY.

Schank, R., and Abelson, R. 1977. *Scripts Plans Goals and Understanding, An Inquiry into Human Knowledge Structures.* Lawrence Erlbaum Associates Publishers, Hillsdale, NJ.

Schmolze, J., and Brachman R. 1982. *Proceedings of the KL-One Workshop.* Bolt Beranek and Newman Inc.

Schriber, T. 1972. *A GPSS Primer.* The University of Michigan Ann Arbor, Industry Program of the College of Engineering, IP-841.

Shannon, R., and Mayer, R. 1986. *Models and Artificial Intelligence.* Submitted for publication in Knowledge Based Modeling and Simulation Methods, North Holland Press.

Shannon, R., Mayer, R., and Phillips, D. 1986. Knowledge Based Simulation Techniques for Manufacturing. *Proceedings of the SME AI in Manufacturing Conference.* (Longbeach, CA.). SME, Dearborn, MI

Sharvy, R. 1966. *Logic: An Outline.* Littlefield, Adams, and Co., Totowa, NJ.

Simmons, R. F., and Slocum, J. 1972 Generating English discourse from Semantic Networks. *Proceedings of the Seventeenth Meeting of the Association for Computational Linguistics* Stanford, CA. (August).

Soloway, E. et al. 1982. MENO-II: An AI-Based Programming Tutor. Research Report #258. Department of Computer Science, Yale University, New Haven, CT.

Sowa, J. 1983. *Conceptual Structures: Information Processing in Mind and Machine.* Addison-Wesley Publishing Company, Reading, MA.

Stefik, M. 1981. Planning with Constraints (MOLGEN: Part 1). *Artificial Intelligence* 16. pp. 111 – 140.

Stefik, M., Bowbrow, D., Mittal, S., and Conway, L. 1983. Knowledge Programming in LOOPS: Report on an Experimental Course. *The AI Magazine.* Volume 4, No. 3.

Stefik, M., and Bowbrow, D. 1986. Object-Oriented Programming: Themes and Variations. *The AI Magazine.* Vol 6, No. 4.

Sterle, M., Snow, P., Wheeler, J., Mayer, R. J. 1986. Weed Control Advisor for Rice Production Technical Report. Rice Producers Association, Beaumont, TX.

Stoy, J.E. 1985. *Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory.* MIT Press, Cambridge, MA.

Sutherland, I. E. 1963. Sketchpad: A Man Machine Graphical Communication System. Ph.D. Dissertation, Department of Electrical Engineering, MIT, Cambridge, MA.

Symbolics Inc. 1986. *Programming the User Interface.* Symbolics Inc. Cambridge, MA.

Teichroew, D.,and Hershey, E. A. 1977. PSL/PSA: A Computer-Aided Technique for Structured Documentation and Analysis of Information Processing Systems. *IEEE Transactions on Software Engineering,* Vol. SE-3, No. 1, pp. 25 – 30.

Teitelman, W. and Masinter, L. 1981. The InterLisp Programming Environment. *IEEE Computer.* pp. 25 – 33.

Turner, R. 1984. *Logics for Artificial Intelligence.* Ellis Horwood Limited, Chichester, UK.

Waters, R. C. 1984. KBEmacs: A Step Toward the Programmer's Apprentice. *M.I.T. AI Laboratory Technical Report No. 753.* M.I.T., Cambridge, MA.

Weizenbaum, J. 1966. ELIZA *Communications of the ACM*, 9, pp. 36 – 45.

Wilensky, R. 1983. *Planning and Understanding, A Computational Approach to Human Reasoning.* Addison-Wesley Publishing Company, Reading, MA.

Wilkens, D. 1984. Domain Independent Planning. *Artificial Intelligence.* Vol 22, pp. 269 – 301.

Winograd, T. 1983. *Language as a Cognitive Process.* Addison-Wesley Publishing Company, Reading MA.

Woods, W. A. 1970. Transition Network Grammars for natural language analysis. *Communication of the ACM*, 13:10, pp.30 – 36.

Woods, W. 1975. What's in a Link: Foundations for Semantic Networks. *Representation and Understanding, Studies in Cognitive Science.* Bobrow, D. and Collins, A. (editors), Academic Press, Inc. New York, NY.

Yngve, V. H. A. 1962. Random Generation of English Sentences. In *The 1961 Conference on Machine Translation of Languages and Applied Language Analysis.* Her Majesty's Stationary Office, London.

Zeigler, B. 1975. *Theory of Modeling and Simulation.* Addison-Wesley, Reading MA.

Zeigler, B. 1984a. *Multifaceted Modeling and Discrete Event Simulation.* Academic Press, New York, NY.

Zeigler, B. 1984b. Multifaceted Modeling Methodology: Grappling with the Irreducible Complexity of Systems. *Behavioral Science.* Volume 29.

Zeigler, B. 1984c. System-Theoretic Representation of Simulation Models. *IIE Transactions.* Volume 16, No. 1, March.