COGNITIVE SKILLS IN MODELING AND SIMULATION

Volume I

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 I

A Dissertation

by

RICHARD J. MAYER

Approved as to style and content by:

December 1988

ABSTRACT

Cognitive Skills in Modeling and Simulation
(December 1988)

Richard J. Mayer, B.S., Purdue University, West Lafayette, Indiana
M.S., Purdue University, West Lafayette, Indiana
Chair of Advisory Committee: Dr. Don T. Phillips

Major advances in simulation techniques have resulted from refinements to our understanding of the modeling and analysis processes. Previous work has provided the framework for advances in the three major support technologies of (1) algorithms, (2) data structures, and (3) statistical methods. The emergence of theories in knowledge acquisition and reasoning from the domain of Artificial Intelligence (AI) provide new methods for study of the cognitive processes of the customer and the simulation analyst within a decision scenario framework. The availability of software and hardware tools for implementing these theories provide a promising mechanism for the construction of advanced modeling systems based on the results of relevant models and theories related to the cognitive process.

This research establishes a base from which an intelligent, model based, systems simulation environment can be constructed. The ultimate goal of such a system is to partially or totally replace the existing human systems simulation analyst. This dissertation will address the underlying theories and strategies of how those cognitive processes can be represented and supported in a knowledge based environment.

To Paula
Simon
Anna

ACKNOWLEDGMENTS

It would be impossible to list all of the individuals who made possible this work. I would particularly like to acknowledge the members of my committee. This dissertation could not have been accomplished without the assistance of my committee chairman Dr. Don T. Phillips. Don has been a long term associate, mentor, and friend. Without his guidance and support, such a broad and complex research program could never have been tackled, much less finished. Dr. Leland T. Blank led me in the establishment of the Knowledge Based Systems (KBS) Laboratory and provided support for this research effort. Dr. Peter Sharpe's enthusiasm for the topics covered in this research provided the impetus for me to complete this work. Dr. Don Friesen introduced me to the areas of formal languages and artificial intelligence and served as a model of professionalism in these fields. Dr. Guy Bailey assisted me with the natural language problems and initial ontology issues associated with this work. Dr. Marty Newcomb expertly guided this work through to completion.

I would also like to acknowledge the many other professional associates at Texas A&M who directly contributed to the completion of this work. I would like to extend my appreciation for the professional, organizational, and personal support provided by Dr. G. Kemble Bennett, Head of the Industrial Engineering Department. I would also like to acknowledge the commitment and support provided by Dr. Herbert H. Richardson, Dean of Engineering, and the Texas Engineering Experiment Station which made possible the Knowledge Based Systems Laboratory environment which was critical to the accomplishment of this research. Without the patient tutorage and contributions of Dr. Chris Menzel, none of the difficult portions of the following work could have ever been realized. Without the help of Ms. Paula S.D. Mayer, neither the cognitive nor mechanical components of this research could have been performed. I would also like to thank Ms. Patricia G. Friel for the countless hours of review and discussion which transformed disjointed ideas into a somewhat contiguous whole. I would like to acknowledge Dr. Robert Shannon who contributed greatly to the initial structuring of the problem areas through our collaborations on several key publications. I should also note that without the example and inspiration provided by Dr. Brian Deuermeyer, I could never have charted such a unique path myself. Finally, I would like to thank the entire faculty and staff of the Department of Industrial Engineering who accepted me as a colleague and whose friendship and sacrifices on my behalf made this effort possible.

Much of the intuition and experience upon which this research was initiated resulted from my previous work with the United States Air Force Integrated Computer Aided Manufacturing program. I would like to acknowledge the contributions of the professionals who guided me during those formulative years and who have continued to provide encouragement, support, and desperately needed technical advice during this research effort. Therefore, I would like to acknowledge Dr. Robert Brown, Stu Coleman, Dr. Tom Cullinane, Sam Nusinow, Reuben Jones, Dr. Alan Pritsker, Mr. Nate Tupper, and Dr. Vincent Russo.

I would like to particularily acknowledge Mr. Timothy Ramey who contributed to many of the ideas and concepts during the development of this research. My sincerest hope is that one day Tim is afforded a similar opportunity as I have enjoyed so that he can set straight the errors and oversights which I have unwittingly introduced in my interpretation of those ideas.

I would also like to extend my appreciation to Nick Berstein, Paul Condit, and Mark Hoffman of the United States Air Force who provided support for much of the formalization and methodology components of this research. I would also like to acknowledge Dr. Ken Morrison, Dr. Frank Plonka, Joe Bulat, and William Knappenberger of Chrysler Motors who believed in the potential for this work and provided funding to support its realization. I would like to thank Mr. Phil Walker for his assistance in collecting system descriptions as well as for his concepts and ideas and for serving as my expert knowledge source.

Finally, I would like to acknowledge the contributions of the research staff of the Knowledge Based Systems Laboratory who were responsible for transforming loosely defined concepts and notions into working systems. The KBS Laboratory was initially conceived as an environment where students and researchers could gather together and share ideas, experience, and resources. Pat Friel, Paula

Mayer, Murali Krishnamurthi, Al Underbrink and I recognized the need for such an environment as we were planning our respective research programs in 1984. The bottom line (as should become evident to any reader willing to plough through this thesis) is that in order to investigate the currently interesting set of cognitive skills in modeling, design, language understanding, or mechanism fault reasoning requires a broad knowledge of many different areas within the Artificial Intelligence domain. Without an institution such as the KBS lab, this type of research would be difficult if not impossible to perform. I certainly know that without the constant technical and moral support of this initial group, my work would never have come to fruition.

In the body of this dissertation I have made every attempt to recognize contributions from the other researchers in the KBS Laboratory which include: Sue Wells, Mark Sterle, John Morris, Jeff Lockledge, Sherri Messmer, Jackie Wheeler, Paul Squitteri, Tom Blinn, Mike Futrell, Janice Jordan, Lou Decker, Keith Ackley, Steve Cook, Min-Jin Lin, Arthur Keen, Chaun-Jun Su, Chun-Jack Huang, Les Sanders, Uday Reddy, Joel Toland, Howard Brodale, Charles Bodenmiller, Terri Layton, Paul Collier, and Rhonda Carter. Each of these individuals have made the work not only intellectually stimulating, but also personnally rewarding.

I would like to thank my parents, Raymond and Patricia, for the opportunities they provided thoughout my life. Every accomplishment is built on previous foundations. Through their sacrifices the foundations for this accomplishment were laid. Also I wish to acknowledge my brothers and sisters who provided me the acceptance and emotional support necessary to take on this endeavor.

I must acknowledge the constant assistance, support, love, and sacrifice from my children, Simon and Anna, that have made the last five years possible. This work can not make up for all that I missed in their daily routines – soccer games, school meetings, and social events, and I know that I could not have done it without them.

Finally, I must acknowledge the support, love, and companionship provided by wife Paula. Without her encouragement, I would not have started much less completed this effort.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

1. INTRODUCTION AND RESEARCH OBJECTIVES

"It sometimes happens in science that the vocabulary of a particular theory becomes so ingrained that the scientist starts confusing the empirical data with its theory-laden description." [Barwise and Perry 1983].

1.1 Research Goals

The goals of this research are fourfold:

1. To advance the understanding of the processes of:
   1.1. How a domain expert becomes aware of how the system in his domain functions,

   1.2. How the domain expert identifies problems in that system,

   1.3. How solution concepts are formulated,

   1.4. How modeling and analysis of models are used in the problem solving process.
2. To establish a system which can assist in this discovery, design, and model building process (hereafter referred to as the Knowledge Acquisition and Modeling Support System KAMSS),
3. To identify the major theoretic and method needs of such a system, and to develop the critical concepts to fill those voids,
4. To develop prototypes of the key components of the theory, methods, and the system concept required to illustrate viability of the developed concepts.

Thus, it is by design that this research addresses the totality of a complex system. The purpose was to establish a framework with the necessary motivations, design structures, theoretic concepts, methods, and proof of viability prototypes. This framework has served as a mechanism for structuring the development of

---

This dissertation follows the style and format of the journal ACM Computing Surveys.

the pieces of the KAMSS for application in manufacturing, engineering, and integrated information system development domains.

1.2 Research Objectives

Relative to the above stated goals the following objectives were established:

1. Perform the required data collection and knowledge acquisition required to establish the basis from which to analyze the requisite cognitive processes.

2. Characterize the cognitive processes involved in the acquisition of an understanding of a system, the use of that understanding in problem identification, solution design, and modeling.

3. Define the requirements for a KAMSS based on that characterization and on a needs analysis performed with actual end users of such a system (manufacturing engineers, systems analysts, and simulation modelers).

4. Develop an architectural design of a KAMSS which satisfies the stated requirements and which satisfies the design goals of:

   4.1. Integration with existing engineering and manufacturing information systems,

   4.2. Provision of Intelligent Assistant support to both the domain experts and the simulation modelers.

5. Establish the basic concepts and theoretical foundations for the knowledge bases and reasoning methods required for development of such a system.

6. Construct relevant methods and prototypes which provide a basic technology for construction of the eventual system and demonstrate the feasibility of the concepts proposed.

1.3 Background

The impetus behind the research reported in this dissertation stems from experiences of the author over the past ten years in the application of simulation and modeling to manufacturing systems. This experience also includes participation in the development of languages, tools, and environments to support the use of modeling and simulation in manufacturing system development and operations decision analysis. A recurring phenomenon during this period was the persistence of the semantic gap which existed between the way the customer understood his environment and the models and simulations which were constructed to assist that person in making decisions within that environment. Another frustrating fact was that the decision makers were never quite able to actually use the languages, tools, and environments which were constructed to provide them access to the modeling and simulation technology. Even after many man-years of development effort, the manufacturing decision makers continued to be dependent upon the availability of the human simulation analyst for access to that technology.

One of the reasons behind this breakdown in technology implementation is the semantic gap between the concepts and tools used in the modeling and simulation process and those used in the common sense reasoning of the manufacturing decision maker. Another reason is the gap between the common sense notion of simulation and the mathematically based simulation theory which is supported by the available tools. In Section 2, we will present a set of hypotheses proposing that the primary view of (and the need for) simulation by a non-simulation analyst is one of *reasoning about the structure of a system*, or *reasoning about implications* of a proposed change to a system. The difference between the two forms is significant and complex. On the mathematical simulation side, we are primarily dealing with models which are an approximation to the truth (known to be false but close!) On the reasoning side we are dealing with an abstraction of the truth (known to be incomplete but true!) [Kuipers and Patil 1987]. Both forms are required; however, the absence of automated support for the latter has impeded the acceptance and usefulness of the former.

The importance of this form of deductive simulation extends beyond the realm of improved management decision making. To be successful (as well as safe) the complex design and manufacturing systems of the future must have the capability to fall back on a knowledge base of reasoning capabilities which include domain common sense, qualitative, and causal reasoning methods. They must be capable of understanding how unplanned behavior comes about and the limitations of the resources they have control for responding to those situations. This kind of capability requires the representation and manipulation of qualitative information about the physical and organizational systems within their purview. The concepts, formalizations, and methods investigated in the course of this research can be directly applied to the provision of these types of capabilities in future systems. [1]

As another illustration of this point it is seldom that someone (other than a simulation analyst) describes an operation in terms of "activity nodes, queue nodes, select nodes, or activity cycles." These are only a few of the modeling concepts (transformed into programming mechanisms) which have come into use over the past twenty years. The recent generation of simulation languages have added additional syntactic sugar to try to bridge the gap between the customer and the simulation technology with little success. The problem stems from the fact that current simulation language constructs are focused on providing additional ease for the specification of a simulation model implementation design.

[1] Recently I was engaged in a conversation with a plant engineer in a large petroleum refinery during which the point was made that the simulation analyses that the industrial engineers performed were generally useless. The position was defended by relating incidents in which the consequences of a change were not taken into consideration by the analysts. In one of the examples, the "IEs" were proposing the replacement of a hazardous waste disposal system which used 55 gallon drums with one which used 500 gallon containers. The simulation apparently demonstrated remarkable savings in handling, acquisition, and transportation costs. Unfortunately, (as was delightfully pointed out) "no one ever considered whether the floors in the plant could support the new containers and associated handling equipment." What was interesting about the discussion is that the plant engineer had *expected* the simulation to "tell them to check that out." He expected that the simulation modeling system would have the capability to analyze a proposed design in a manner which included reasoning about the logical consequences of the proposed change. While I immediately defended the honor of my elected profession, I could see the reasonableness of his argument. The theoretical design and implementation requirements for providing the kind of reasoning capability in a modeling and analysis support environment which that plant engineer expected to see is the major focus of this dissertation. So, in a manner of speaking, this dissertation is an investigation into the consequences of attempting to add reasoning capabilities to a modeling and simulation support environment.

Unfortunately there is a complex set of system analysis, analysis planning, and model design activities which must precede model implementation. There has been no simulation modeling system developed which supports these additional activities.

1.3.1 New Paradigms for Simulation

Major advances in simulation techniques have in the past resulted from refinements to our understanding of the modeling and analysis process. Previous work has provided the framework for the application of advances in the three major support technologies of:

1. Algorithms,
2. Data structures,
3. Statistical methods.

The emergence of theories in knowledge acquisition, semantics, and reasoning from the domain of Artificial Intelligence (AI) provide new methods for study of the cognitive processes of the customer and the simulation analyst within a decision scenario framework. The availability of software and hardware tools for implementing these theories provides a potential mechanism for the construction of advanced modeling systems based on the results of relevant models and theories related to the cognitive process. One of the objectives of this dissertation is to characterize the cognitive processes of the customer and analyst within a decision scenario framework. This characterization is presented in Section 2.

In this document the terms "customer" and "domain expert" are used to refer to the manufacturing decision maker who is assumed to be experienced in his own discipline but is not considered to be trained in simulation modeling or system analysis. The term "analyst" is used to refer to a person trained in the use of simulation modeling and analysis. The analyst may or may not have expertise in the manufacturing domain where he is applying his analytical skills. However, it will be pointed out in Section 2 that often the most effective analysts for a particular problem area are those who have experience with solving problems in that area (i.e. an analyst with experience in material handling system design will often be more effective in the construction of simulation models in that domain than an analyst without such experience).

The objective of this research is to establish a base from which an intelligent environment can be constructed for:

1. Capture of a system description,
2. Automatic model generation,
3. Qualitative and quantitative simulation,
4. Deductive problem analysis,
5. Problem solution recommendation.

The ultimate goal of such a system is to partially or totally replace the existing human systems simulation analyst. This dissertation will address the underlying theories and strategies of how the following cognitive processes can be represented and supported in a knowledge based environment. Our focus is to examine how:

1. The customer understands and operates within his manufacturing system,
2. The customer recognizes symptoms or formulates concerns,
3. The customer analyzes his environment and identifies the causes of symptoms,
4. The customer designs solutions to these problems,
5. The customer uses models to aid in the analysis and solution design tasks,
6. The customer comes to the recognition that a simulation analysis can support his decision making or analysis process (including investigation into the common sense notion of simulation),
7. The customer constructs goals for simulation analysis,
8. The customer communicates an understanding of the system (the system description) to the systems simulation analyst,

9. The customer communicates a need for information (goals for analysis) to the systems simulation analyst,

10. The analyst understands the system description and customer needs,

11. The analyst determines the requirements for the model and the necessary simulation analysis (particularly how the analyst uses past experience to anticipate unstated customer needs),

12. The analyst determines the model architecture and experimental approach which will meet the analysis requirements,

13. The analyst uses experience with a particular set of modeling constructs to build a detailed specification of the model design,

14. The analyst uses the results of the statistical analysis of the simulation output data to meet the original information requests (or goals) of the customer.

The concepts presented here are unique from both the view of the simulation discipline and from the AI perspective. The uniqueness from the AI point of view arises from the fact that the process under study involves the interaction of natural skills (i.e., the ability of a person to describe the dynamics of his own environment) and learned skills (i.e., the ability of the systems analyst to conceptualize, design, and build usable computer simulations from the customer's description). The characterization of the cognitive processes provided in Section 2 forced the evolution of the knowledge based systems design presented in Section 3. The requirements of this architecture led to the formulation of a new theory of knowledge acquisition semantics and reasoning presented in Sections 5 and 6. The intended contribution of this work to the modeling and simulation disciplines comes from the focus on the modeling of common sense reasoning about system dynamics as the basis for design of an advanced modeling and simulation environment.

Current simulation languages have evolved to support the efficient programming and execution of a simulation model. Thus, they are able to contribute to the decision making process only in the actual analysis phase. The support of the other phases of the decision making process has awaited the characterization of the actual thought processes which occur in these phases and the development of the representation and reasoning capabilities required to emulate / support the activities in those phases. The concepts presented in Section 2 provide that characterization and have allowed us to design a knowledge based environment which will be able to provide support of all phases of the decision making process. This support spans:

1. Acquisition of descriptions of the engineering or manufacturing environment,
2. Representation of the semantic content of those descriptions in a form which supports:
   - 2.1. Tailoring of the description acquisition mechanism,
   - 2.2. Incremental expansion of the underlying ontology mechanisms,
   - 2.3. Model generation,
   - 2.4. Specialized domain reasoning,
   - 2.5. Application generation support.
3. Model and analysis application generation,
4. Data connectivity to existing company information systems,
5. Experimental design and execution,
6. Results interpretation.

1.3.2 Summary of Previous Research

Fundamentally, there are four major approaches to the application of AI methods to the activities associated with modeling and simulation analysis.

1. The first approach would be to utilize the methods and theories of AI to model the cognitive processes associated with model building, model based decision making, and simulation analysis.

2. The second approach is the emulation of the common sense reasoning about the causal and qualitative aspects of the system structure and dynamics.

3. The third is to combine quantitative simulation models from a specific domain with expert systems constructed for human experts in that domain.

4. The fourth approach is focused on utilizing existing programming languages and tools developed to support AI investigations or expert system application developments to construct simulation models in a particular analysis arena.

With respect to the first approach, the most significant work to-date appears to be that of George Heidorn [Heidorn 1972]. Heidorn's Natural Language Processing System (NLPS) supports the development of queuing system simulations in GPSS from natural language input. In an interactive session the user describes a problem situation to NLPS. Through the use of a set of encoding and decoding rules the NLPS constructs an internal representation (IPS) of the described situation in the concepts of a queuing system model and hence is able to query the user for additional information which is required to make that model complete. When a complete model is developed, the user can request that a simulation model be constructed. The system generates a model based on the IPS and queries the user for such experimental parameters as the length of run. The system was developed as a study in natural language processing, not as a study in modeling support environments. However, it provides a landmark demonstration for many of the concepts which will be developed in this dissertation including:

1. Interactive user dialog to describe a user's problem,

2. Ability to store a problem description and answer questions about that problem description (limited to queuing model domain of discourse),

3. Ability to generate a simulation implementation from a problem description.

Limitations of the NLPS include:

1. Use of queuing model semantics as the basis for internal representation which limits the type of descriptions which can be captured,

2. Lack of a capability for the user to input the goals for analysis which means that rather than design a particular model the NLPS generates a standard model,

3. The natural language processing paradigm is based on a multi-level pattern matching approach which is dependent on a complete lexicon which means that for practical usage the system would have to be greatly expanded.

Of the other literature reviewed to date, relative to the use of Artificial Intelligence / Expert System (AI / ES) methods in the system modeling and simulation domain, only Gaines [Gaines and Shaw 1985] appears to recognize the basic difference between the concept of a simulation model and the models of cognition embodied in an ES. However, he falls short of actually identifying the need to perform the tasks implicit in the first approach. Neither does he report on any actual work performed in this area. McArthur [McArthur 1981] makes reference to attempting to accomplish the first task, but only in a passing comment to the fact that object oriented programming supports a notion of "egocentric" modeling. I believe the failure to recognize the potential for application of AI methods to simulation support system design lies somewhere in the tendency of engineers to want to view programming methods strictly as mechanistic tools for the construction of products, rather than as a medium for experimenting with concepts about how people think (a domain largely in the past relegated to students of psychology and philosophy).

The work reported in [Elmaghraby and Jangannathan 1985] could be classified in the first area, except that in this description of a simulation language selection system under construction there is no mention of analyzing the planning strategies of the expert, nor of any attempt to assist the end user with the characterization of a particular application. This work appears to be entirely property driven using a deterministic procedural set of decision logic (more appropriate to a group technology decision table based stratagem).

McRoberts in [McRoberts et al. 1985] describes ongoing work in the study of generation of model abstractions from system descriptions. However he fails to recognize that the emulation (or even study) of the human process in modeling has any merit. Therefore, he falls back on traditional deduction techniques based on general statistical and systems theoretic methods and presumes as a starting point an existing "detailed model which is free from run time errors." This appears to contradict the initial hypothesis that customers can describe their systems, but need help in the development of simulation models of those systems, since it presumes a simulation model as the starting point of his method. It is interesting to note that he does admit failure with all but the simplest of reduction approaches.

Work along the lines of the second approach most closely parallels the focus of this dissertation. This work includes research in the areas of:

1. Causal reasoning (e.g., qualitative physics [Davis and Brown 1984; Kuipers 1984]),
2. Common sense reasoning about physical systems (e.g., naive physics [Hayes 1979; Hobbs 1985]),
3. Continuous process qualitative reasoning (e.g., qualitative process theory [Forbus 1984; Kuipers 1984]),
4. Planning and mechanical system design (e.g., reasoning about plans [McDermott 1985]; common sense planning and understanding [Wilensky 1983]),
5. General theory of natural language semantics (e.g., based on situated agents [Barwise and Perry 1983] for continuous concepts [Bunt 1985]).

Each of these works addresses the problem of common sense reasoning about the causal and qualitative aspects of system structure and dynamics from either natural language descriptions of the system or from common sense models of those systems. The work by Bunt, Hayes, and Barwise focuses primarily on representational issues as addressed in Section 5 of this dissertation. The work by McDermott and Wilensky focuses on the planning issues which we found were required for generation of qualitative models from the descriptions. Finally, the work by De Kleer, Forbus, and Kuipers addresses the problem of reasoning with qualitative models formed from these representations.

Relative to the third approach, O'Keefe provides an interesting taxonomy for all the possible uses of an ES with a simulation model (i.e., ES's inside simulations,

simulations inside ES's, ES's on the front of simulations, ES's on the rear of simulations, ES's loosely coupled to simulations, ES's and simulations as a part of something bigger, etc.) [O'Keefe 1986]. To a large extent, this article reflects a general naive understanding which many simulation modeling experts appear to have (at least as indicated by their literature) relative to what AI technology is all about. Several of the statements made in this article are so absurd that they deserve comment. The opening sentences of the paper claim that, "Simulation and expert systems are remarkably similar. Both employ various representations to model some aspect of an uncertain world, with the model being formed as a piece of computer software." Of course, the fact that what constitutes a model, what is being modeled, the criteria for interpretation, and that the criteria for acceptability are altogether different in the two disciplines is completely overlooked.

The paper also seems to confuse the notion of "rules" as a method for encoding knowledge with "flow of control" language constructs that utilize "IF – THEN" syntax. In several other instances, the author equates implication and computation with inference, (i.e., "What makes simulation and expert systems methods similar is that they are each based on a modular representation of a system, with an inference mechanism that drives this representation." and "Representation within simulation include events, activities, process interaction, and differential equations, where the inference mechanism for the first three is a next-event time-scheduling algorithm, and the subsequent is an equation solving time-slicing method.")

Possibly the most misleading claim of this paper is that "Rather than test an expert system on a user or a real environment, the expert system can be tested on a simulation." Considering that such a test would require that the simulation embody not only the complete knowledge of the expert in order to evaluate the responses of the ES, but also a complete enough understanding of the problem to generate a representative sample of problems, makes one question the rationale for constructing the ES in the first place. At least all of the knowledge which was supposed to be encoded in the ES would have to be present in the simulation in order to test the ES!

Reddy [Reddy et al 1985] identifies the task of automating the "instrumentation of a simulation model" as a domain within which an ES could profitably be employed. The general notion proposed is that a customer should be able to specify a system performance goal (note that this is different from an analysis goal) and the ES would automatically instrument the model to collect the appropriate data needed to generate the system performance measures. He provides an excellent characterization of the various types of performance data which an ES would be required to instrument the model to collect. However, he has not actually built the proposed ES nor does he address the reasoning process which would have to be captured in such an ES. Finally he limits his attention to only numeric based performance measures and does not presume that the ES would have any knowledge of the application domain.

The work reported in [Baskaran and Reddy 1984; Cleary et al 1985; Reilly et al. 1985; Futo 1984; Futo 1985; Adelsberger and Neumann 1985; Adelsberger et al. 1985; Ko and Wheeler 1983; Shannon et al. 1986; Mayer 1986] falls into the fourth category of investigations – the use of existing AI/ES programming methods to attack the development of more effective simulation modeling environments. The work written in Prolog reported in [Adelsberger and Neumann 1985; Adelsberger et al. 1985; Futo 1985; Futo 1984] is interesting in that they propose the representation of simulation goals as Prolog goals with the idea that a Prolog interpreter could be used to automatically perform the necessary backtracking to achieve the simulation goal. This approach is interesting for two reasons. First, it assumes that the notion of a simulation goal is the same as the head of a Horn clause. Secondly, it would lead one to believe that "intelligent backtracking" strategies exist which could alleviate the obvious combinatorial search problems. However, the discovery of such strategies is a major unresolved research issue in the logic programming community [Campbell 1984; Lloyd 1984].

1.4 Statement of Major Hypotheses

In pursuit of the goals of this research, the following tenets have guided the investigation to-date, and hence form the basis of the approach taken:

1. There is disparity between the way humans understand and reason about system dynamics and the formal reasoning methods embodied in the underlying mathematical theory of modeling and simulation (see [Bobrow and Collins 1975; Schank 1982; Wilensky 1983; Dyer 1983; Harman 1986; Turner 1984] versus [Corynen 1975; Mesarovic and Takahara 1975; Ziegler 1975; Ziegler 1984b, Pritsker 1977]). This incompatibility limits the usefulness of existing methods both from the point of view of what can be modeled and of what use can be made of the models which are constructed. Other formal bases for reasoning can be expected to give rise to completely new methods of simulation analysis which can be applied in semantic domains and which reduce to the traditional simulation methods as a special case.

2. The modeler is communicating in a world of representations, while the customer is dealing in a world of descriptions. Thus, for a modeling and simulation support environment to be successful, it must provide support for the capture of both system descriptions and model designs. Such an environment must provide for tracking the allocation of the system description elements to the model elements which occurs during the model design. The modeling environment must also provide for the augmentation of the model design representation with the rationale for the choice of model structure or model component specification. One of the goals of this study was to define a structure and representation for "descriptions" as separate from "models". Another was to show that development of analysis tools which operate on the such descriptions can accelerate the provision of useful results to the decision maker and in many cases eliminate altogether the need to actually perform quantitative simulation.

3. The simulation model generation process is primarily an engineering design process and not a system description process. The analyst uses a system description and a question (or set of questions) to be answered to decide on the level of abstraction, the boundaries and the components of a model based upon the data he needs to extract from an experiment. That model, within a particular modeling environment is then used as the experimental medium. Thus, just as a chemist designs an apparatus to establish and control an environment for the examination of the properties of a chemical system, the simulation analyst designs a model architecture and components

to support his experimentation with a particular manufacturing system (real or imagined). It is conjectured that failure to recognize this fact is a major reason behind the communication problems which exist between the simulation modeler and the customer. Assuming this view also sheds light on the unsolved problem of simulation model reuse. To achieve effective economies of reuse of the analysis investment we must find a way to capture and store the system description itself, and largely automate the model design based on that description and new questions to be answered.

4. The design process is primarily an inductive process. That is, the human designer guesses a solution based on experience and then adjusts that solution to the particular problem requirements. This process paradigm is assumed to hold both at the level of design of solutions to a particular manufacturing problem and also at the level of design of the model and reasoning or analysis technique being used to understand the manufacturing problem. Thus, if the goal is to combine the simulation analysis capability with the potential for goal seeking behavior (e.g., self-modifying design behavior), then the system must include the capability to perform at least element driven design [Mayer 1985]. This must be supported with an ability to decompose defined system elements into components. Such behavior is possible, but the major limiting factor is the inability of the system to recognize when a component level solution is reached (i.e., to embody the system with the capability to "know when something can be built") [Ramey 1983]. To achieve this level of design capability requires a more complete representation of "the way the world is" than is achievable in current analytic models. This led to the study of semantics of natural language and information flow as a model of the primary human reasoning process.

5. There are two major approaches to the process of constructing a simulation based analysis plan. These two approaches can be profitably combined. The first, will be characterized as *entity tracing* [Pritsker and Pegden 1979; Phillips 1979; Pegden 1982] the second as *condition / trigger tracing* [Deshler 1981; Markowitz et al. 1978; Kiviat et al. 1973]. *Entity tracing* is most commonly found in applications which involve the study of population behavior, and has been the traditional method for manufacturing applications. *Condition / trigger tracing* is most commonly found in applications involving the

study of complex logic systems, and has been the traditional method used in software and procedural system analysis. Considering a target quantitative simulation language with both capabilities allows for the automation of the quantitative simulation model design from qualitative models extracted directly from the system descriptions.

1.4.1 Methodological (Conceptual) Contributions to AI Foundations

Because of the relative early stage of AI concepts as a scientific paradigm, AI approaches tend to fit poorly into the traditional "experiment driven scientific method". This problem is recognized by the AI community and activities are constantly under way to formulate realistic guidelines for AI research. The following excerpt from Ritchie [1984] provides insight into the ways information is structured in this dissertation:

"There are various ways in which the typical AI project (e.g., a doctoral thesis) could contribute to our collective knowledge:

1. It could introduce, in outline, a novel (or partly novel) idea or set of ideas. For example, ideas such as 'consider a semantic structure as a computational procedure', or 'regard computation as a sequence of messages between autonomous entities'.

2. It could elaborate the details of some approach. That is, starting from some idea, the research could criticize it, or fill in further details, in order to transform a slogan-like idea or metaphor into a fuller theory. This activity is comparable to theory-construction in a traditional science, but it is not directly tied to some standard means of testing, as will be discussed below.

3. It could apply some articulated theory to some actual subject matter or data, and report the consequences. This is the nearest counterpart to traditional "experiment", but it differs in some important respects. Typically, the practical investigation proceeds by writing and running a computer program, and the assessment of the result of this activity is difficult, since AI ideas tend not to be formulated in such a way as to allow specific success / failure judgments."

1.4.2 AI Hypothesis Testing

Relative to the last type of contribution, Ritchie goes on to suggest that "an empirical AI 'experiment' can be scrutinized in various ways:

1. Experiment design (static): Does the structure of the program reflect the theory it purports to test?
2. Experiment design (dynamic): Does the internal processing behavior of the program correspond to the dynamic aspects of the theory (if any)?
3. Consistency: Has the program been successfully run (repeatedly)?
4. Results: What were the consequences, in terms of internal behavior and, in terms of output, of the program's execution?
5. Interpretation: What do these results mean in terms of the theoretical constructs?
6. Conclusions: What are the consequences for the theory of the interpreted results?"

The evaluation of our hypotheses should also be based on their ability to contribute to an improved theory of the cognitive processes involved in modeling, analysis planning, and system understanding. The value of such improvements to the research and industrial communities must be evaluated based on their contribution to an improved theory of simulation and the utility of existing simulation analysis methods in practical applications [Overstreet and Nance 1985].

1.5 Approach and Products

The approach followed in the course of this research built upon our experience in knowledge based systems research and development as well as experience in the methods for large scale integrated information systems development. Since the KAMSS spans both these domains the process required continual revision and tuning. Thus, the following steps reflect a summarization of the "should be" approach rather than a documentation of the "as was" approach.

1. Perform knowledge acquisition and knowledge engineering for the process identified in the process characterization.

2. Perform analysis of written manufacturing and engineering system descriptions.

3. Develop ontology for domain concepts.

4. Define user requirements for support.

5. Define functional requirements for the KAMSS.

6. Define KAMSS system architecture.

7. Examine the role of natural language processing technology for the KAMSS architecture.

8. Develop a method for semantic theory evolution. Illustrate that method by application to Situation Semantics to show how that theory could be extended to accommodate the needs of KAMSS.

9. Develop a model of reasoning sufficiently robust to account for the types of reasoning support required of the KAMSS.

10. Determine the impact of the characterization of the cognitive processes, the semantic theory, and reasoning models on the quantitative simulation model design process. Illustrate how these products enable the development of a qualitative simulation (or causal reasoning) capability to augment the quantitative modeling process.

11. Develop prototypes of the various components (and component methods) of the KAMSS needed to establish viability of the design and the underlying principles.

Each of these tasks required the performance of literature reviews, examination of the state of the art, identification of voids, development of hypothesis, design of solutions based on the hypothesis, and testing or evaluation of the solutions either via prototype or argument. It should be noted that the "system" orientation of this effort led to a pragmatic approach to each of the tasks. Thus, rather than attempt to develop from scratch (particularly in the theory oriented areas) the focus was on selection of something close and modification to suit the needs.

1.5.1 Research Products

Using the above described approach the research objectives were met through the production of the following products:

1. The characterization of the system understanding, analysis and modeling process provided in Section 2 from the domain expert and system analyst points of view,
2. The KAMSS requirements and architectural design presented in Section 3,
3. A methodology for discovery and organization of a domain ontology presented in Section 4,
4. A strategy for natural language processing and generation which embodies both application of an existing ontology and the extension of that ontology,
5. A methodology for semantic theory development,
6. An application of that method for semantic theory development to the extension of situation semantics to encompass the base ontology components required to support description acquisition and reasoning in the KAMSS environment,
7. A representation scheme for capturing "descriptions",
8. A reasoning method (based on establishment of chains of information flow) flexible enough to support both the description acquisition, modeling, and qualitative reasoning requirements of KAMSS,
9. A reasoning method using tokenization and constraint propagation for discrete qualitative simulation,
10. A methodology for modeling ontology elements (IDEF1/ES),
11. A methodology for capture of process flow and object state descriptions (IDEF3),

12. An object based simulation language development environment which supports both the event tracing and condition trigger tracing paradigms,

13. A prototype system description capture environment,

14. A prototype model development support environment,

15. A prototype qualitative simulation environment.

1.6 Organization of the Dissertation

The remainder of this dissertation is logically organized into three major parts (eight sections). The first part presents a characterization of the thought processes involved in decision making based on the use of models and simulation (Section 2), and the conceptual design of a Knowledge Acquisition and Modeling Support System (KAMSS)(Section 3). The second part (Sections 4 through 7) examines the basic theoretical foundations required to realize the KAMSS architecture. The third part (Sections 8 and 9) describes the proof of engineering prototypes which were constructed to demonstrate particular elements of the KAMSS concept (or as tools which would be needed to construct an actual KAMSS) and the conclusions of this research effort.

1.6.1 Summary of Section Contents

The results of the research directed at the objectives stated above are presented in the remaining sections of this dissertation. The following paragraphs provide an overview to the contents of each section.

Section 2 is an introspection of the thought processes associated with the assimilation of an understanding of a manufacturing system, the construction of models of that system and the performance of simulation analysis based on those models. Section 3 describes an architecture for what amounts to the automation of the analyst function in the support of this process. This architecture is provided for the purpose of illustrating the kind of automation support which would logically follow from an understanding of the cognitive processes described in Section 2.

Sections 4 through 8 contain results needed to establish the architectural viability of the concepts presented in the KAMSS architecture. Section 4 describes the natural language processing technology required to support the user interaction features of the KAMSS. The linguistic theory underlying the discourse processing and generation which the KAMSS must perform is reviewed. A methodology is presented for the development of domain specific ontologies. This methodology allows application of the KAMSS to domains other than engineering and manufacturing. Section 5 delineates a theory of semantics serving as the basis for representation of the system description information as well as the model design information. Section 6 describes a set of reasoning strategies required to support the inductive reasoning and deductive logic mechanisms needed to interact with the user during the information acquisition process, as well as to build and manipulate the respective models. This section elaborates the concepts presented in Section 5 to explain the process of understanding and describing system dynamics. It focuses on how this understanding is used in common sense causality determination processes.

Section 7 compares and contrasts the mathematical definition of a model which underlies current simulation practice with the customer's representation of a system. This representation is what the customer considers as a model of his system. This section explores the issues of the differences between these two representations and outlines the abstraction techniques for generating various models consistent with the customer understanding of his system. Finally this section describes a technique for using common sense reasoning as a simulation process. The benefits of such a capability over traditional mathematical modeling techniques is described. Section 8 describes the various prototypes which were constructed for the purpose of engineering proof of concept systems. The final section summarizes the findings of this research, overviews the problems of scale-up of the prototypes and outlines areas for future research.

2. CHARACTERIZATION OF COGNITIVE PROCESSES

The goal of this section is to provide a characterization of the cognitive processes involved in decision making based on the use of models and simulation. The purpose of this characterization is two-fold. First, it provides the answers to the questions posed about the cognitive processes of the customer and the analyst. Second, the characterizations are used as requirements for the KAMSS architecture described in Section 3 and the data necessary to support the semantic theory and reasoning strategies presented in Sections 5 and 6.

2.1 Overview of the Process

The process of making decisions based on the use of models and simulation analysis is depicted as a cyclic process in Figure 2.1. The process involves two major roles: that characterized as the *customer role* and that referred to as the *system simulation analyst role*. The decision making process starts with observations made on the existing (or planned) environment, which subsequently give rise to the recognition of known failings in the existing system (symptoms) or perceived possible failings (concerns). The process taken by a customer to determine the underlying cause of these observations leads to the determination that simulation might provide a vehicle for validating a suspected cause/effect relationship or determining the performance characteristics of a desired change. At this point the customer usually engages the services of the system simulation analyst. The precise nature of the interaction, and what is communicated to the analyst, is dependent on the type of decision process with which the customer is involved (e.g. situation explanation, problem cause identification, design generation, design analysis, etc.).

The analyst (or group of analysts) generally performs an evaluation of the customer's system, desired analysis results, and system description provided by the customer. This evaluation is combined with the "goals" of the customer and results in a set of requirements for the simulation analysis. This set of requirements is used as a basis for the process of model and experiment design. The process of model design is viewed as a two step process, with the first step responsible for the definition of the model structure and components, and the second step

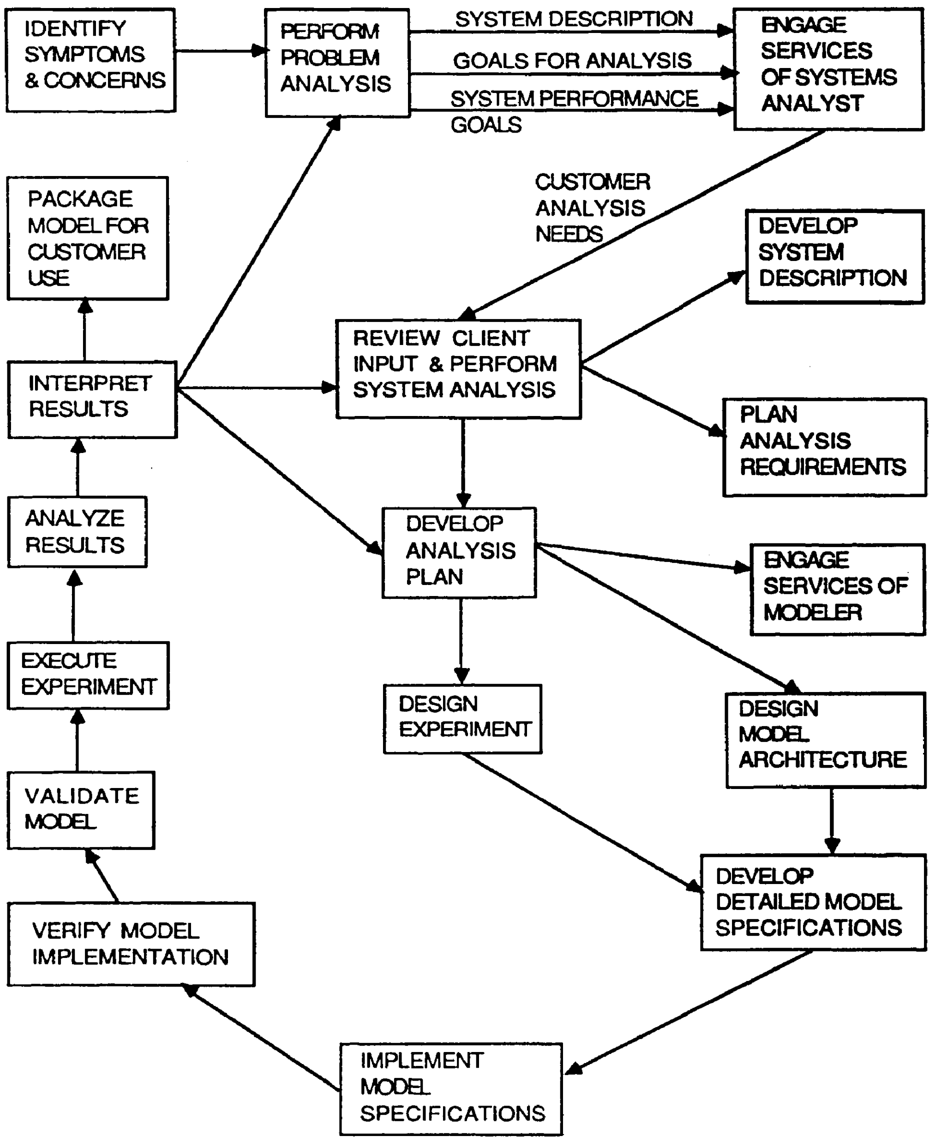

IDENTIFY SYMPTOMS & CONCERNS
PERFORM PROBLEM ANALYSIS
SYSTEM DESCRIPTION
GOALS FOR ANALYSIS
SYSTEM PERFORMANCE GOALS
ENGAGE SERVICES OF SYSTEMS ANALYST
CUSTOMER ANALYSIS NEEDS
PACKAGE MODEL FOR CUSTOMER USE
DEVELOP SYSTEM DESCRIPTION
REVIEW CLIENT INPUT & PERFORM SYSTEM ANALYSIS
INTERPRET RESULTS
PLAN ANALYSIS REQUIREMENTS
ANALYZE RESULTS
DEVELOP ANALYSIS PLAN
ENGAGE SERVICES OF MODELER
EXECUTE EXPERIMENT
DESIGN EXPERIMENT
DESIGN MODEL ARCHITECTURE
VALIDATE MODEL
DEVELOP DETAILED MODEL SPECIFICATIONS
VERIFY MODEL IMPLEMENTATION
IMPLEMENT MODEL SPECIFICATIONS

FIGURE 2.1: DECISION MAKING BASED ON MODELING AND SIMULATION ANALYSIS.

responsible for the detailing of the characteristics of the components and the system interfaces [Mayer 1985; Mayer and Young 1984; Ramey 1983]. The process of experiment design parallels the process of model design, and may in fact be inseparable from that process [Corynen 1975; Zeigler 1984a].

Depending on the implementation method used, the construction of an executable version of the model can be a separate activity or it can be a side effect of the detailed design specification. Most of the research to date on simulation language design has focused on improving the power and usability of the detailed design model specification language structures which are used in the programming process [Mayer and Young 1984; Mayer 1986; Overstreet and Nance 1985; Shannon et al. 1986]. A part of the construction process is the verification of the functionality and behavior of the programmed model. It is interesting to note that there has been little work reported in the area of language development to support the specification of the experimental design. It is also interesting to note that the verification phase of the simulation process rarely includes a detailed requirement for either the verification or the validation of the experimental design. An important consideration of the construction process is the development of data interfaces with existing factory information systems. These interfaces are often required to allow the execution of the desired experiments in a timely and cost effective manner.

The next major step in the simulation analysis process involves the validation of the programmed model and a comparison of the experimental model against observations on the real system. This step is frequently overlooked either because sufficient "real world" data is too costly to obtain or because the "real world" system does not yet exist. In both cases (and even in the cases where validation experiments are performed) the actual validation is normally performed via expert consensus that there is a "reasonable" (supportable) argument for the hypothesis that the model actually represents the behavior of the existing or proposed system. This latter form of validation in fact is often referred to as the "qualitative" analysis.

Following a validation process, the analyst executes the planned experiment which generally involves executing multiple "runs" of the simulation program collecting both raw data in the form of traces and summarized data in the form of plots, histograms, and/or tabulated data. The analyst then performs a statistical or logical analysis of that data and synthesizes a picture of the predicted behavior of the modeled system. This process is often referred to as "results interpretation". It should, in fact, be called "results analysis" because the next step taken by the analyst is to *use* these statistical analysis results and interpret or "understand" them in the context of the original goals which the customer identified. This process may cause the redefinition, and reinitialization of the preceding steps. That is, the customer may decide that simulation analysis is not the type of analysis really needed. The analyst may decide that the analysis requirements were incorrect or both might decide that the model or experimental design must be modified.

The following sections provide a description of the major cognitive processes involved in each of the above described tasks and the relationship of current AI theories of similar cognitive processes.

2.2 Perception of Systems

The intent of this section is to investigate the question of "*How does a person come to know about and understand a manufacturing environment?*" We will first discuss the indicators of such cognitive processes by an examination of the way people describe their manufacturing situation. We will then discuss considerations which arise by examination of previously proposed [Wilensky 1983] theories of common sense planning and situation understanding. We conclude that the human is attuned to certain uniformities of property values which extend over space and time. These uniformities allow him to carve up the system into individual parts of the system which stand in relation to other parts of the system in order to achieve an order or structure. We will also conclude that part of the perception process involves developing a common sense theory of system dynamics. This theory (which includes common sense notions of cause and effect) is used in many of the decision making steps outlined in the previous section.

2.2.1 Formulating System Descriptions

When people are asked to describe their manufacturing system, the normal response combines several descriptive mechanisms. Typically they will first describe the product *"These parts range from large cylindrical rollers to precision gears* *and cams."* Next they will describe the "physical situation" *The production facility is organized into five fabrication areas feeding the assembly area*, and the "organizational structure". Finally, they will generally construct a series of scenarios which define a sequence of events causally linked, precedence sequenced in time, or procedurally linked by the operational policy of the organization and normally focused on the product. It is often difficult to formulate a single scenario which captures the *entire collection* or "sequences of events". One reason for this difficulty is that the interaction between the scenarios is either indirect or because the situations which are being described are realized in parallel. Normally the dialog will focus on a single thread of activities or events corresponding to a particular organizational viewpoint which the person believes provides a basis for the justification of a particular hypothesis. Typically, the description is augmented with a graphical interpretation such as that illustrated in Figure 2.2. These illustrations are used during the discourse as a way for the customer to essentially transport his "resource" situation (knowledge of the way the world is [Barwise and Perry 1983]) to the location of the discussion.

There are at least two different types of event ordering which are typically included in these system descriptions; the first can be characterized as "arrival ordering" and the second as "activation ordering". Activation ordering of events implies a binding between events which is assumed to be causal in nature (ie. the occurrence of event α precipitates the occurrence of events β, γ, etc.) *"Before the ARAC can begin to function, there are several setup activities required."*. Arrival ordering merely implies a relation between events (ie. a condition for event β to occur is that event α has occurred) *"The Material Planner issues a Request for Procurement to the Buyer upon receipt of a work order."*.

The task of formulating and communicating this type of system description is one of structuring the presentation of a path through the *Time, Condition, Object* space illustrated in Figure 2.3. The initial description of the products of the facility provides an effective basis for the initial communication of the "Object" portion of this space. By describing the state transitions of these objects over time a framework for the "Conditions" portion of this space is established. This framework can then be exploited as a mechanism for structuring the descriptions of the activities which are known (or believed) to provide the causal basis for

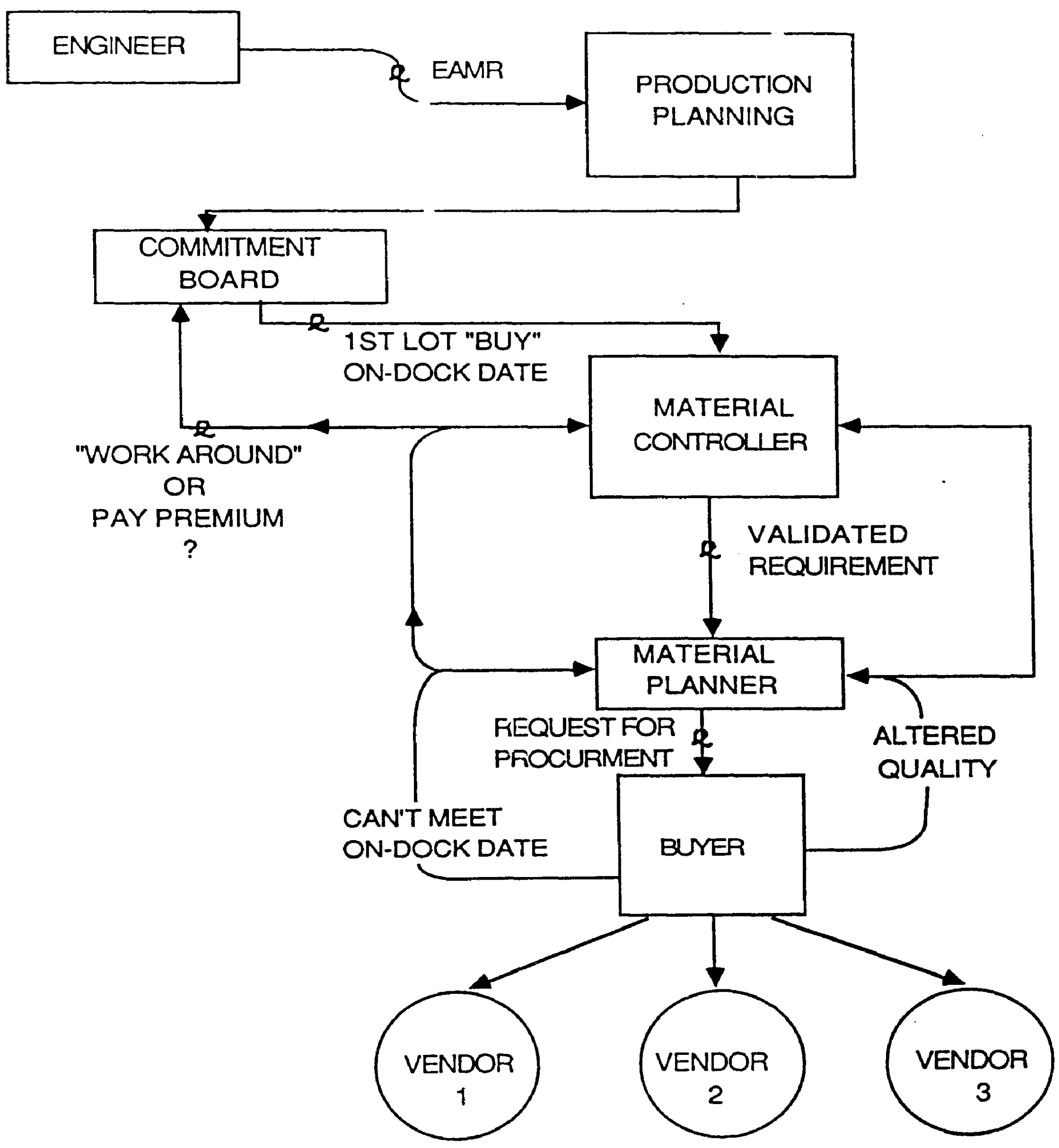


FIGURE 2.2: TYPICAL SYSTEM DESCRIPTION SKETCH.

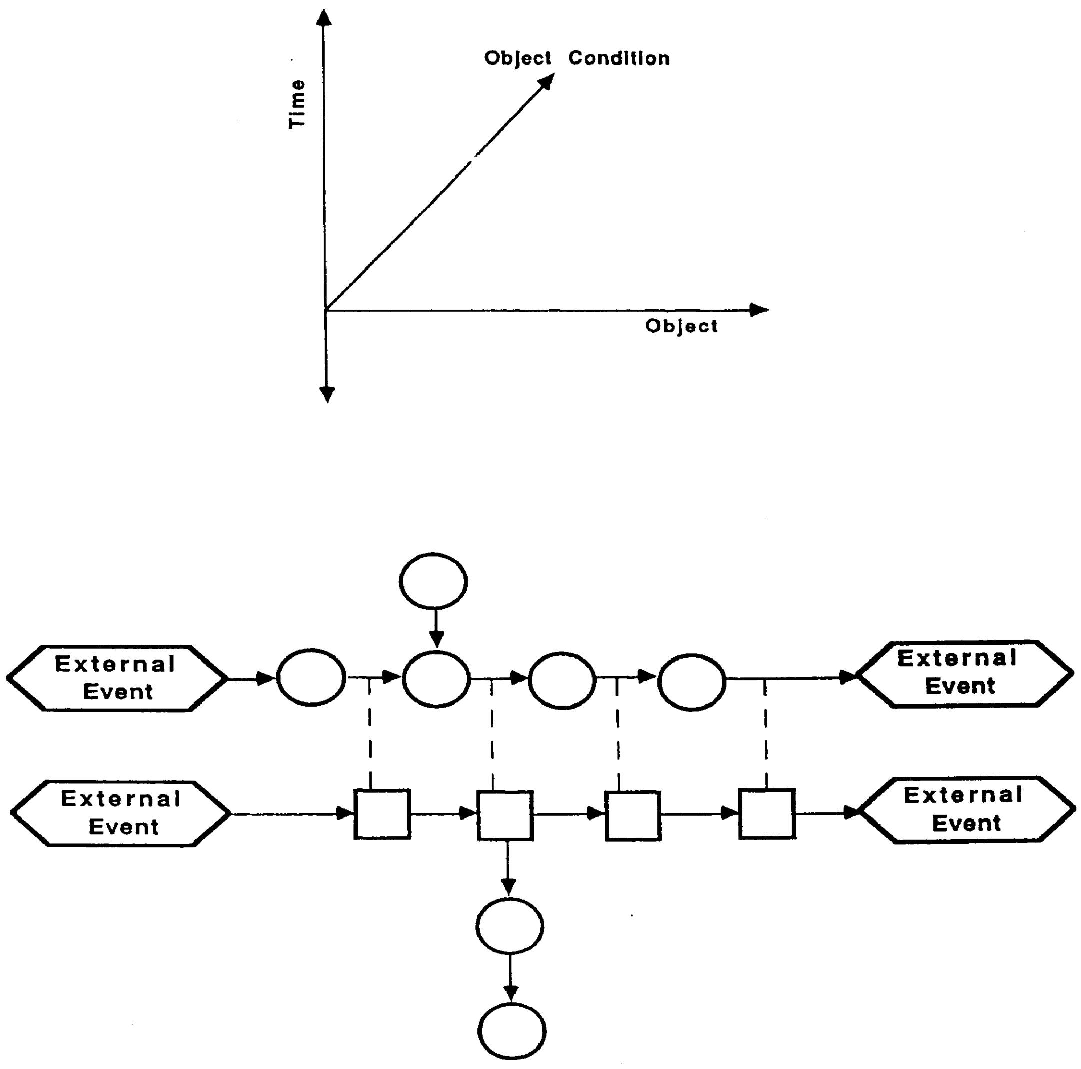


FIGURE 2.3: INTERACTION OF OBJECTS, TIME, AND OBJECT CONDITION.

the state changes. The sketch illustrated in Figure 2.2 is illustrative of a path through this space using an activity orientation.

One of the basic differences between the type of system descriptions of manufacturing systems, versus electrical [Davis 1984] or physical [Forbus 1984] systems, is the focus of manufacturing descriptions on the "procedural rationale" which links activities. For example *"EAMRs are created by Engineering at the conceptual and preliminary design stage of a new program in order to allow time for Material to procure long lead time items..."*. These constructs are the primary indicators of the understanding that the speaker has about the organization system dynamics. In a very real sense these constructs convey the manufacturing "logic" as understood by the speaker. Language constructs which typically indicate this type of "procedural rational" include subordinating prepositions such as:

1. As required to...
2. According to...
3. In order to...
4. To account for...
5. For the purpose of...
6. Due to...

as well as situation constraint representations formulated in terms of the familiar "If - Then, and When - Then" structures.

One important observation which can be made on the manner in which people describe manufacturing systems is that the descriptions are almost always incomplete. That is, a single individual rarely is encountered who can give a uniform description of the entire system. The boundaries of the knowledge of the individual are usually marked with a specific declaration of uncertainty or a radical change in level of detail in the description and a corresponding lack of causal rationale attributed to the elements of the description. Another important observation which can be made about the perception of systems which will become relevant in the later discussions about providing automated support for modeling is that the individual is *a part of the system*. Thus internalization of the knowledge acquired about the system will certainly occur for the portion of the system functions which the individual performs. This implies a lack of objectivity that will likely handicap the individual in recalling, describing, and explaining the perceptions which have been developed.

Finally, review of actual system description texts (as well as drawing on our experience with interviewing manufacturing engineers, supervisors, and management decision makers) indicates that, in many (if not most) situations, objects are referred to not by a "name" but rather by some sort of definite description. For example:

1. The lathe in the machine shop,
2. Long lead time items,
3. A problem EAMR,
4. The loading robot,
5. The shop supervisor.

This tendency is more prevalent in the description of a system than in the actual operational discourse of the environment. The tendency to use definite descriptions is driven by the need to express patterns of behavior and rules. That is, one of the roles that definitive descriptions can conveniently serve is as a variable which can be incorporated into conditional structures to express rules of behavior or explanations. The definite descriptions provide the language mechanism for generalization. For example:

1. '*Items not meeting specified requirements because of incomplete operations* are routed through the rapid rework areas.'
2. 'When *semi-processed units* are received, they are checked by the *material receiving department personnel.*'
3. '*The rough and semi-finished parts procured from vendors* are stored on racks.'

What can be hypothesized from the above observations is that the perception of a system is built upon the following mechanisms:

1. Perception of collections of properties which maintain a uniformity over time with respect to their value assignments and co-locational occurrence, (These uniformities are the system parts.)

2. Perception of interrelationship of parts,

3. Associations of the collection of parts with a particular set of tasks, which these parts (standing in a particular relation) support,

4. Associations of the collection of parts with a particular problem, which these parts (standing in a particular relation) attempt to solve,

5. Development or acquisition of a theory of system dynamics which can be used to explain the behavior (transformations / state changes) of the system over time.

This perception of a system is quite complex. It includes an organization of perceptions built up over time, as well as perceptions extending through time. If we consider the original psychological notion of *schema* as a network of concepts [Sowa 1983], then the concept of a "system" can be considered as a special type of schemata. The process of perceiving a system then can be characterized as the process of building a particular schema or set of schemas over time. If we consider the original notion of a *script* as a framework for organizing series of events [Schank and Abelson 1977], then the development of a concept of system dynamics can be considered as a process of the development of a set of these script frames with the appropriate schema attached. It can thus be argued that the process of describing a system is synonymous with the process of perceiving the system, in that it forces the individual to discern the elements of the system, identify them as unique, and construct a scenario for describing their interactions over time. The scenario has many of the characteristics of a plan. It can be viewed as a collection of assertions each of which provides a description of a state of affairs or a state of change. However, rather than being a partial simulation of the future (as one would find in a plan) it is an explanation of the past. Thus the assertions serve as the vehicle for explanation. If an assertion is presented as the effect of an action, then that action is considered the causal explanation of the assertion. If on the other hand the assertion is true at the initial state of the scenario then the causal explanation is ascribed to the environment of the system and relative to the system description it is considered as a premise.

The description of a system then is considered to have three parts:

1. The description of the parts,
2. The scenario(s) executed by the system in its operation,
3. The function of the parts (i.e. the role played by the parts in the scenario(s)).

To understand how such descriptions are acquired and used in a problem solving environment we need to look more closely at the issues involved in common sense planning and situation understanding which is the focus of the next section.

2.2.2 Planning and Understanding

Another way that we can hope to learn about how people develop knowledge and understanding of a manufacturing system is to examine previous work in common sense planning and understanding. Much of the success of AI/ES programs to date have been on problem-solving programs which start with a well defined initial state of the world and which proceed to search a well defined solution space to achieve a well defined goal [Mayer 1986; Friel and Mayer 1985]. These types of cognitive activities predominate many of the engineering and manufacturing technical activities. However they do not accurately characterize the more predominant activities of the workplace. As pointed out [Barber 1982; Wilensky 1983] the goals of organization based systems are vague, the information relevant to achieving the goals is not clear and there is more than one individual working (sometimes in conflict) to achieve the perceived goal. The perception of systems and their problems in these situations is a result of the activities of understanding the work situation, and using common sense planning to accomplish goals within that environment. Thus, while the system can be viewed as a problem solving mechanism, the individual within that system is more than a passive component

[Ramey 1983]. The individual is an intelligent agent which attempts to understand his role and will contribute over time to the modification of the system itself.

The above implies that a person's perception of a system is intimately linked to the daily activities of common sense planning and understanding. Through the process of inferring the existence of a goal from the observance of actions, the individual develops the information needed to explain the existence of the goal. This explanation becomes the basis for the formation of the system concept. The individual must also use this understanding of the process to infer his own goals based upon knowledge of the mission of the system and the current situation. The planning process uses these goals to guide the structuring of a sequence of actions to achieve the goals. The explanation structure which was generated during the understanding process is used as an important source of constraints for the planning mechanism and also as a mechanism for the evaluation of the "acceptability" of a plan. But much more information is needed to construct a plan. Part of that information is knowledge of physical and organizational dynamics (i.e. common sense knowledge of cause/effect relations which can be used as the basis for action postulation). Another important piece of information is knowledge of past plan failures which provide templates for guiding the structure of the plan [Dyer 1983]. This information is used in a continuous planning process which involves the following activities [Wilensky 1983]:

1. Goal Detection,
2. Plan Proposing,
3. Plan Projection,
4. Execution,

It is in analysis of the daily implementation of the first three of the above described set of activities that we can postulate how a person acquires his knowledge of the system and develops perceptions of symptoms and concerns. This analysis can also identify characteristics of the support environment which must be provided to support the customer in his situation analysis process.

2.2.2.1 Goal Detection

One of the important elements of this process is the recognition that something has changed in the environment or in the system itself. The recognition process may be restricted to a change in a property of the system already recognized (the level of in-process inventory is 26 weeks). Or it may be a recognition of the emergence of a new property as being important to the description of an existing object (the inventory level is rising), or the deletion of an old property from consideration. Finally it may involve the recognition of the emergence of new object instances (as in the arrival of a new order) or in the emergence of a new class of object types into the individual's perception. The emergence of these new perceptions causes the formulation of percepts and the association of these percepts to existing concepts or the addition of new concepts [Sowa 1983]. Since these new phenomena must fit within the existing perception of the system one of the sources of symptoms and concerns is the inability of the individual to explain the role of these noticed phenomena in his existing "theory" of the system. Such problems become "important" or "interesting" if they affect the use of that theory in the plan formulation activity of the individual.

Another important aspect of the noticing activity comes about by the fact that the planning and understanding processes run continuously and in parallel. This means that the noticing activity is monitored in the context of a known plan and its projection. A mismatch between the anticipated outcome of a plan or any part of the plan is grounds for the recognition of a plan failure. It is in the attempt of the individual to explain why a plan failed that we come across our second common sense notion of system dynamics as will be explained in Section 2.2.3. The fact that all discrepancies or plan failures do not result in the recognition of a symptom or concern can be explained by the notion of Theme Abstraction Units (TAU) introduced in [Dyer 1983]. Dyer proposes the existence of these TAUs as a sort of condensed experience base for how plans can fail in an uncertain environment. Thus, for example, taking an action prior to plan projection is an example of the "look before you leap" TAU. Dyer proposes that the collection of *adages* of a culture (which would include a specific work environment) are used by the individual both as a mechanism for evaluation of plans during the plan formulation or projection process and also as a mechanism for explaining the failure of plans. Dyers proposal is useful for explaining how one is able to perceive a "problem" in a system. Based on his ideas we can postulate that, it is only when we cannot explain a noticed failure of a plan via one of these TAUs that we accumulate evidence of a problem in the system.

2.2.2.2 Plan Proposing

As Wilensky describes this process [Wilensky 1983], plan proposing is the act of deciding on a network of tasks which will accomplish a perceived goal. His view of the planning process focuses on the existence of a set of goals for the planning process (called meta-goals) and plans for achieving these goals (called meta-goals). The meta-goals are potentially organized into themes that are situations under which particular meta-goals should be pursued. The highest level meta-themes include:

1. Don't waste resources,
2. Achieve as many goals as possible,
3. Maximize the value of the goals achieved,
4. Avoid impossible goals.

These meta-themes are particularly relevant to our particular concern. If they accurately characterize the common sense reasoning processes involved in planning and understanding in the daily work environment, then we can conclude that a major part of the concern of the manufacturing decision maker is the balancing and tradeoff between these meta themes. This balancing requires several types of knowledge. The first is knowledge of the "things" (i.e. resources) needed to attain a goal. Resources can be characterized as:

1. Time,
2. Consumable objects,
3. Non-consumable objects,
4. Abilities,
5. States (e.g. locational proximities).

The second meta-theme implies the ability of the decision maker to recognize both inconsistencies between goals and subsumption relations between goals (i.e. know when the achievement of one goal entails achievement of another). Finally it identifies the fact that goals can occur in two different manners, either as the result of the recognition of a theme situation, or as the result of attempting to satisfy a precondition of a task plan.

The impact of Wilensky's goal driven multilevel planning paradigm is particularly important to the design of the explanation generation capability in KAMSS. From this theory we can see that when the request for an explanation of an event requires an item which could be reasoned as a logical consequence of a plan, then the meta-theme rational for the plan itself is generally what the customer would expect the system to produce as an explanation.

2.2.2.3 Plan Projection

The notion of a capability to generate possible worlds which reflect conditions which would exist after a plan has executed, we believe to be at the heart of the common sense notion of simulation. In considering what actually happens (from a reasoning point of view) during this plan projection process, we get a clear picture of the differences between this process and the capabilities of traditional mathematical simulation. One important difference is that the traditional model of simulation (as proposed by Zeigler [Zeigler 1984a]) presupposes the existence of a set of interacting processes. On the other hand consider yourself sitting in front of the word processor creating this text (as I happened to be doing some time ago). As you consider the task at hand and the possible outcomes of performing this task, you project an interpretation of the words and concepts that you are formulating as they will be interpreted by the readers of the document. You then envision the opinions which will be formulated and the likely actions which will result from those opinions. You are constructing possible situations, evaluating the desirability of those situations and constructing/validating a plan which will presumably result in an acceptable outcome. In fact it is only after you have done such a projection that you have a complete enough task network to consider classification of some of those tasks as measurable processes which could be formulated into a simulation model.

2.2.3 Reasoning With Common Sense Theories of System Dynamics

Describing the cause and effect of noticed changes is the basic focus of system dynamics. Individuals construct "theories" of system dynamics to support the perception, planning, and understanding of tasks previously described. The term system dynamics is chosen for the purpose of this discussion not to refer to the classical input/output notion of mathematical theories of system behavior, but rather to refer to the collection of physical, organizational, and logical cause and effect explanations for noticed phenomena in each of these areas. Ken Forbus identifies three properties which a theory of dynamics must possess if that theory is to be useful in supporting common sense reasoning about "physical" systems [Forbus 1984]. First it must specify direct effects and the means by which effects are propagated. Second, it must provide some means of handling the problem of decomposability (i.e. describing the behavior of a complex system in terms of the behavior of some decomposition of that system into parts). Finally it must allow for graceful extensions. On this last point he elaborates to include a notion of monotonicity in deduction on observations which (unfortunately) causes problems outside of the world of physical phenomena. This point will be taken up in more detail in Section 6. We add another important characteristic - that a theory of common sense reasoning about system dynamics must be learnable. That is, one of the quality measures of any such theory would be the reasonableness of acquisition of the ontology, domain knowledge and reasoning methods. As the next section will describe, these common sense theories are central to the process of planning and understanding of the individual's environment, therefore they must be acquired by each individual.

The important issue at this point is that individuals use internalized theories of system dynamics to assist in the planning and understanding processes. The following is a discussion of some of the cognitive tasks which are supported using these common sense models of reasoning.

1. *Situation Understanding:* Situation understanding refers to the process of inferring goals from observations of actions. This process relies on acquired theories of system dynamics and is a possible source of new theories.

2. *Situation Anticipation:* One of the characteristics of "intelligent" objects is the ability to project a current situation into the future, predict a new situation, and affect current behavior as though the predicted situation was actual. This process is different from the plan projection task (described below) in the reasoning mechanisms involved (conditional constraint formulation), and in the fact that it is performed in the absence of any definite delineation of the actions which will lead to the future state. Situation anticipation plays a major role in the generation of "concerns" described above, since concerns are generally considered to be perceived failings (or shortcomings) of an existing system based upon an anticipated future world. Thus, for example, the manufacturing engineer may believe that the material handling system is inadequate based on his anticipation for a major influx of new business and not on actual shop load.

3. *Plan Projection:* The deduction of future situations from a current situation based directly on the cause/effect structures of the theory of system dynamics being used, the perceived situation, and the formulated plan of actions. In performing plan projection the human applies the believed cause/effect relationships to the current situation and a formulated plan in order to predict what a future situation will be like after the actual execution of the plan.

4. *Post Mortem Analysis:* The construction of an explanation of how a particular situation, state of affairs, or course of events has come about. This explanation normally attempts to show that the particular phenomena could be "deduced" from a set of universal laws applied to the set of objects and relations in the system, starting from a given initial state.

5. *Consistency Analysis:* Judgment of the reasonableness, acceptability or consistency of the results of any of the above tasks. For example, the evaluation of the reasonableness of the assumptions made or the soundness of the reasoning process used. Generally used in a critical or comprehension mode.

6. *Observation Interpretation:* Deduction of what is happening (i.e. what is the situation) and the individuals, objects, and actions involved from a partial set of observations.

7. *Experiment Planning:* Based on a hypothesis and/or observation interpretations, "experiment planning" is planning of required actions or situations which will allow the collection of observations needed to support or disprove the hypothesis. Experiment planning requires an underlying concept of the "inner workings" of the system to know what aspects of the system are observable and how to design mechanisms for collecting these observations.

8. *Causal Reasoning:* Formulation of a structure of descriptions asserted as reasons for the existence of known facts. This is one of the most typical forms of common sense reasoning based on an understanding of the system dynamics. Unfortunately it is often confused with deductive argument — the latter providing only a method for giving reasons for believing a fact.

One important point which can be made about the above forms of reasoning, which use common sense theories of system dynamics, is that each of the above represents a type of "simulation" in the common sense notion of the term. The fact that these reasoning tasks are generally performed with the assistance of the analyst (during the *problem definition phase* of an analysis effort) is what gives rise to the common phenomena of the customer being satisfied with the services of the analyst prior to the model ever being constructed. In fact, it is common for simulation projects to fail because the analyst does not get the opportunity (or does not take the time) to "communicate" these reasoning acts with the actual decision maker. What happens in these situations is that the customer claims he is "unaware" of the "assumptions" of the model when in fact he is not concerned with the assumptions at all. Rather, he is disappointed by not having the opportunity to acquire a better "theory of his system dynamics" for future use in decision making.

2.3 Identification of Symptoms and Concerns

The second question posed in Section 1 was related to how the customer comes to recognize (or becomes aware of) problems in his environment. This section discusses the process of recognition of symptoms and concerns, and considers how the above presented theories of common sense planning, understanding, and reasoning with theories of system dynamics can provide some insight into this process.

In the "Guidebook to Systems Development" [Ramey 1983] "symptoms" are defined as active failings of a system, "concerns" are defined as perceived failings of a system. There are three possible causes of symptoms. The first is that the task or problem has changed. The second is that the individual's perception of the system has evolved to the point at which the original framework no longer can account for the observed phenomena. The third is that the underlying mechanisms causing the observed phenomena have changed. It should be recognized that at least the first two of these phenomena can occur without the recognition by the individual of a "symptom". For example the change of problem may be merely a change in focus, in which case the original system is still considered to exist as an independent, subordinate, or cooperating system to the one which addresses the current problem focus. For example, we may consider the material handling system as a system in its own right. Simultaneously we may consider it to be a subsystem of the shop work control system if the physical or operational constraints of the material handling system are used to enforce job sequencing. The second reason is closely associated with the formulation of the system concept in the first place, and can be considered to be a part of the process of learning about a system.

One of the ways in which individuals use their perceptions of systems is in the activities of "understanding their environment" and "planning to achieve goals." When (in this continual process of planning and understanding) the level of mismatch between anticipated observations and actual observations becomes significant enough to disrupt our ability to understand "what is going on" or to effectively "carry out our plans" then we classify the disparity as a symptom.

2.4 Performing Problem Analysis

Up to this point we have postulated processes for understanding the manufacturing situation and recognizing symptoms or concerns in that environment. When an individual is faced with a number of these irritating facts an attempt is generally made to ascribe a reason or cause for them. The collection of facts taken together is the problem. Thus one of the tasks is to decide which facts should be grouped together. This is referred to as problem identification. Once the problem has been identified, the formulation of a hypothesis that explains the symptoms by stating their cause is referred to as problem analysis. Up to this point, all tasks previously described can be internalized by the individual. However, the performance of this part of the process in an organization requires communication with the other individuals within that organization. This communication process can be very difficult indeed. For the recognition of the set of facts may not be uniform across individuals. This is particularly true in the case of concerns.

As described above, the perception of both symptoms and concerns presupposes a perception of the system and a knowledge of a theory of dynamics which can be used as the reference point for the recognition. However, it is often the case in all of our experiences that we spend considerable effort bringing others to recognize the systems which appear so obvious to ourselves. If we examine the process of communication between individuals, we can shed some light on the importance of the *problem identification* task and some of the skills required to successfully perform that task.

One of the first tasks of communication associated with the problem identification task is the description of one's perception of what is the system under discussion. One of the problems with communication of a system is that our perception of what constitutes "a system" as described above is a collection of objects which stand in particular relation to one another such that they serve to address a particular problem. In order to communicate such a "binding" without specifically identifying each of the component parts and each of the specific relations one often resorts to searching for a symbol (generally a name or a descriptive reference) which appears to have a connotation which includes the perceptions which we mean to communicate. Once the system description has been communicated then we can consider the listener to share a common "resource" situation. This resource situation is then used to identify the observable phenomena which one considers as the indicator of a failing in the system. Often this process involves convincing the listener that he has seen but has not observed.

Once the observations are 'on the table' then the issue becomes one of evaluating the various "theories of system dynamics" which each participant has relative to the system under discussion. In order for the phenomena observed to be recognized as a "failing" or shortcoming of the system some common agreement of "how" the described system should work must be obtained. This process generally includes each individual attempting to give an "account" of the natural or conventional constraints which he sees to exist between situations and how the observed phenomena fit (or do not fit) into these accounts.

It is also during this process of problem analysis that most all of the forms of non-formal fallacies and language ambiguity raise their ugly heads. Emotive fallacies such as *Ad Populum*, *Ad Hominem*, *Poisoning the Well* and so forth are quite prevalent [Sharvy 1966]. This is one of the reasons why the call for "computer simulation" is often raised. What is being sought is a "neutral" mechanism which can be "programmed" with a theory of dynamics that can then reason through the logical consequences of that theory to explain the observed phenomena or identify a failing in the theory to account for the phenomena.

2.5 Problem Solving

How people actually solve problems and perform design activities is one of the major challenges to researchers in AI. Ramey has observed that the designer does not necessarily deduce the proper system structure from the problem requirements but rather hypothesizes a solution and then uses a form of inductive reasoning to demonstrate that his solution is appropriate [Ramey 1983]. Assuming the validity of the cognitive skills enabled by an internalized theory of system dynamics (described in Section 2.2.3) we can explain a part of this process. Presumably, ***situation understanding, post mortem analysis, and observation interpretation*** have been used in the process of reaching the point of the recognition that a solution is needed. At this point the person is required to postulate a world within which the "problem" would not exist. This is a natural role for the use of the ***situation anticipation*** skill. The question is, "What to focus the application of this skill upon?" If the person focuses on the symptoms or concerns, then we see the classic problem of solutions which treat the symptoms and not the disease (e.g., solving a problem with lateness by increasing the lead time). Alternatively, a person can focus this skill to an underlying theory of system dynamics. This results in situations sometimes characterized as "explaining the problem away". Finally, a person can focus this skill to the "system" its structure, components, and behavior. This focus allows the formulation of conditional constraints which define (presumably) a world in which the system elements which exist or the interaction between the elements which exist are such that the perceived problems do not exist.

2.5.1 Characteristic Driven Design

This type of solution design approach is by far the rarest and certainly the most abstract. In this approach, the designer begins by partitioning the problem requirements based on abstract criteria such as:

1. Common functionality,
2. Common mechanism,
3. Usage patterns.

These partitioned classes are used to define abstract mechanisms which one attempts to match against known characteristics of solution approaches. If no matches are found, the process essentially recurses to a lower level until a match is discovered. As the system elements are identified and matched against characteristics of known solutions, the design structure is examined for part balance and adherence to physical constraints. It is only generally after the major part of the design is completed that the focus is turned to the system interfaces.

2.5.2 External Constraint Driven Design

This approach to systems design is generally used in situations where problem requirements or the problems themselves are poorly formulated. The customer develops solution concepts which start by treating the solution as a black box. Through the construction of usage scenarios, the characteristics which that black box must exhibit are identified. The usage scenarios can be thought of as scenarios in which the black box solves the problems at hand. Once these scenarios are defined, the customer then has a set of solution characteristics which he uses to "partition" the available technologies. That is, he matches the solution characteristics with characteristics of the technologies of which he is aware, allowing him to eliminate large classes of alternatives quickly. Once he has established the "potentially viable" set of solutions, the customer will pick a particular element of the solution and use that selection to "force fit" other parts of the solution. The second and third part of this type of design process is very similar to the process of opportunistic planning with constraint propagation. This type of design approach can be characterized as inside-out design, since the process essentially starts with the selection of a critical system element which forces the form of the rest of the elements until the system interfaces are met.

2.5.3 Element Driven Design

This characterization of the design process appears to be similar to that of the external constraint driven approach except that the process works from the outside in. That is, the designer chooses the pieces of the system which he knows he will need (generally dictated by the presumed system interfaces) and through process or item tracing determines both the functionality and implementation characteristics that the new system must exhibit. This strategy is generally supported by the designer's belief about the relative difficulty of certain parts of the system design. Based on these beliefs, the designer will often initiate the design process around the "most difficult" areas under the presumption that solving the problems in these areas will constrain the remaining areas to the point that they become candidates for a "characteristic driven" design process.

2.5.4 Formulation of Analysis Goals and Model Requirements

According to Minsky [Minsky 1974], "a model is not simply a model; it is a model which can answer certain questions about a certain object for a certain questioner" [Zeigler 1984c]. Systems are not modeled simply for the sake of modeling. The objectives arise from the needs of decision makers to analyze and understand a problem and/or to evaluate and assess the outcomes of contemplated decisions. A contemplated decision may be in the form of a design or physical change in the system, a modified control action, or a new policy or procedure.

The objectives or goals of the user will dictate the appropriate design of the model, the experiment to be run with it, and the analysis to be performed. Such objectives must:

1. Specify the part of the real system which is of interest,

2. The purpose of the analysis,

3. The response or performance variables of interest,

4. The degree of accuracy in measuring the response(s) needed.

Consider the task of symptom identification. The person has information through means of direct observation or communication that does not square with his expectations. Is his perception of the system incorrect or incomplete? Possibly his understanding of the system dynamics is flawed. Any of these might be the case. It is the application of the skills of *post mortem analysis*, and *observation interpretation* which he must use to make this determination. However, the application of these skills often identifies holes in the understanding of the individual. One type of hole can be characterized as lacking information. The individual then may set as his goal for analysis the collection of this missing information. On the other hand, the void may be in the understanding of the system dynamics itself. In this case the goal for analysis may in fact be the extension of his existing theory of the system operation to fill this void.

Alternatively, consider the above described task of problem analysis. It is reasonable to assume that, in the process of attempting to communicate the "observed facts" to other persons inside or even exterior to the organization, the need for corroborative information will be identified. The individual makes use of his *experiment planning* skill to develop a plan for acquiring the information needed to support his story. But in many cases the observations for these experiments are not easily made. For example, what he might like to do is recreate a situation which existed just prior to an observation which gave rise to his recognition of a problem with the plan of instrumenting the situation (possibly with human witnesses) for corroboration. The analysis goals in this situation essentially consist of both the observations desired and the corroboration of these observations (i.e. if you had only been there you would have seen ...).

2.6 Characterization of the Customer/Analyst Discourse

Possibly one of the most important aspects of the interchange between the customer and the analyst is what each perceives the other knows or believes. Another observation which can be made about the interaction between the customer and the analyst is that the customer is often looking for "holes" in his train of thought. This is one of the reasons that a customer will look for an analyst with "experience" in modeling and simulating situations similar to his own. Such experience contributes little to the actual construction capabilities of the analyst; in fact it may degrade them. However such experience generally contributes to the analyst's knowledge of "similar situations" and how those systems worked. Thus the customer is looking for:

1. Alternative explanations of the observed phenomena,

2. Alternative solution or design concepts.

3. Critique of the rationale he has used to arrive at his conclusions,

4. Actual analysis and simulation experimentation.

From his common sense experience, the customer is also aware that he has often been able to figure out what another person knows by "simulating" the other person's reasoning. Therefore, he is confident that if there are inferences which he could make with the facts at hand, but which he has not made for one reason or another, then the analyst armed with the power of "computer *simulation*" will surely be able to find the missing links.

The analyst from his experience will expect that the customer generally does not understand:

1. The simulation methods he was hired to perform,

2. The actual problem situation and the associated cause / effect relations,

3. The implications of the solutions concepts, if any, proposed.

However the mature analyst understands the customer / client relationship, he is generally sophisticated enough to know the limitations of his own knowledge of the actual situation. The mature analyst also is familiar with the "facts" of life in an organizational situation, the typical rivalries and competition between elements of that organization, the flow of power and control, and the way change can be effected to accomplish personal gain. This knowledge allows the analyst to survive, know the relative acceptability of the suggestions he will make, and know the level of confidence he can place in the information provided directly by the customer.

The customer/analyst interaction then proceeds in an iterative fashion in which:

1. The customer provides a partial system description, perceived problems, and poorly formed analysis goals.

2. The analyst attempts to understand the system description, weigh the problems against the evidence provided by the customer, and structure an analysis plan which will result in the data needed to satisfy the analysis goals.

3. The analyst presents a summary of the system description back to the customer for validation.

4. The customer critiques the summary, usually repeating previously provided description and either expanding the description or refining the detail of the description.

2.6.1 Understanding System Descriptions and Customer Needs

One of the problems with understanding the customer's system description is that the customer is generally more aware of the special cases than of the general operation of the system. This is partially due to the fact that the special cases tend to be "noticed" and are less likely to be internalized. Thus one of the main tasks of the analyst is to prompt the user to "stand back" and describe the basic system dynamics. This normally proceeds by getting the customer to describe major events and the arrival or activation causality between events. The following passage is provided to illustrate some of the syntactic and semantic features which characterize a customer's description of his system to an analyst:

OPERATIONS AT JOE'S JOB SHOP

> "In this shop we make the parts we need to repair the presses which we service. These parts range from large cylindrical rollers to precision gears and cams. We turn out 50 to 75 items a week depending on their complexity and who I have working in the shop. Since we specialize in particular presses and have been in this business over 20 years, we have built up a fairly good stock of used parts which we can often refurbish to fill an order. We also have a pretty good handle on what kind of raw stock we need to keep on hand to make the parts we need.

In this shop the average number of operations on a part is eight. We have a senior partner who determines what the operations are and writes them on the production order along with the part number, press identification, quantity, and need date. If the part is a used part we need to clean it in the hot tank first and then mic it before he can plan it out. Our shop foreman is responsible for scheduling the work in the whole area. He generally makes his decisions based on who showed up for work, and what machines are available. Some of the older equipment gives us quite a bit of down time. The foreman sorts through the orders and unless there is a particular 'hot job' he will first try to pick one which is for the same press that the worker just finished a part for. This helps keep the workers busy since he doesn't have to sort through the rack of press specs. Of course if the part has additional operations required and if the machine needed to do those operations is available and if the worker can run that machine then the foreman moves the job to that machine and gets the fellow back to work. We make the worker report to the foreman each time he finishes an operation because we pay on an incentive basis and because hot jobs come in all the time. If we can't do the next operation on the job we put it in the inprocess stores and find another job for the worker to do. We do maintain standard hours for our jobs. This information is on the order, split out by setup and run times. This way the foreman can keep track of how much work is left on a particular job. We have been trying to get the foremen to calculate the the slack time (you know the due date minus todays date minus the amount of remaining processing time) and pick the job with the smallest slack time. However I think most of them do it by feeling. We do keep a running estimate of the amount of work against each machine and if any machine gets more than ten days work scheduled for it we take the low priority non- critical jobs and farm them out.

We run an eight hour shift five days a week unless there is a large overload then we will work overtime. Overtime runs about 10 % each month. We have 20 machines, 15 operators, two foremen three

> machinists, who generally do setups and critical jobs, and three travelers who move work around and do most of the running for odds and ends. We also have two people who manage the raw stock and used parts inventory. They also pack the finished parts for shipment to the mechanics in the field.
>
> Since we were unionized last year we had to hire two dock workers and operate under some rather strict rules. The travelers can only move parts and if the part weighs over 50 lbs they have to use a forklift of which we have only two and one is so large that it can only service two of the machines. The machinists and operators can't move parts and the operators can't set up their own machines. Also the stock room workers can't load or unload trucks nor can they move parts outside of their stock area. The managers can't move the work or setup or run the machines and of course the planners and the rest of us can't order the operators or machinists to do anything! But so far every thing seems to be working out ok.
>
> We've been thinking of replacing two of the older machines with one of (company x's) new NC machines. We know that the start up costs will be significant in training, programming and tape proofing. What we would like to find out is if the improved throughput and reliability will pay off. But we also are concerned whether the systems will fit into our overall operations."

Contrasting this dialog to that typically analyzed in articles/books on natural language understanding [Dyer 1983; Wilensky 1983; Winograd 1983; Schank and Abelson 1977; Allen 1987], linguistics, logic and semantics [Fodor 1977; Barwise and Perry 1983; Mellish 1985] points out many interesting differences. First, the extensive use of first person present tense (e.g., We, I) particularly in the presence of the verb "have". Secondly, there is extensive use of demonstrative pronouns (e.g., this, that, these) as a form of situation reference. The speaker is addressing a listener, not writing a text. The speaker is also often using an "attitude" form of utterance. That is, a form which often presumes that the listener has "seen" or can see the situation which he is describing. In fact many of the sentences could have been preceded by the phrase "as you can see" or "as you saw". This form of utterance is similar to the "seeing" verbs of [Barwise and Perry 1983] which are used to report perceptions and cognition.

Another (possibly not unanticipated) phenomenon is the extensive use of figures of speech such as; "mic", "this way", "by feeling", "overtime runs", "pay off", "turn out". Also much of the text presumes background knowledge of:

1. How manufacturing generally works,
2. How organizational structures work in manufacturing, and how unionization affects an organization,
3. Meaning of specific terms and phrases such as:
   - 3.1. Rollers, gears, and cams,
   - 3.2. Hot tank versus hot jobs,
   - 3.3. Slack time,
   - 3.4. Inprocess stores.

These phenomena are indicative of what might be referred to as a "manufacturing sublanguage" [Sager 1981; Kittredge and Lehrberger 1982; Grishman and Kittredge 1986]. However, from the texts analyzed it is not clear that there is enough consistent structure for such a classification.

The extensive use of "because", "in order to", etc. structures provide explanation directly in the text for the rationale of "why things are the way they are". These structures provide an initial insight into the speakers understanding of his system's dynamics. Other insights into his set of beliefs in this area include the general description of the flow of an item through the shop, the description of the product characteristics, description of the organizational decision making and personnel interactions, and the description of the processing on a typical job.

Another observation which will become important in the design of the description capture environment is the heavy use of plurals and in general mass nouns, particularly in the description of the objects which change over time. The use of word structures like *parts, workers, travelers, etc.* indicate the use of plurals and mass nouns as a language mechanism for describing "general situations" or situation schemas, rather than specific situations. One of the techniques which will become important in the automation of the analysis activities will be the use of these general schemes as frameworks for the generation of specific situation scenarios.

2.7 Formulation of Analysis Requirements

During the interview and analysis process, the analyst will participate in many discussions similar to the one which was characterized in the last section. From information gathered in such discussions presumably one will eventually glean the data required to build a model. In [Zeigler 1984a] a detailed description is supplied concerning the activities associated with system/model specification. However, little or no description is given concerning how one decides what the requirements are for that specification. That is the issue we are concerned with in this section. A "simulation analysis" is a *system development process*. When an analyst is "sizing up a job" or "bidding on a job", or "analyzing a customer's problem," essentially what he is doing is a "needs analysis" and "requirements definition" for the analysis system which he is going to design, construct, and use!

One of the unfortunate handicaps that the simulation analyst often operates under is the presumption (possibly self induced) that the customer has actually engaged his services to employ a mathematical simulation. Thus when he approaches the job he naturally presumes that the solution technology has already been decided upon. This is actually a rare situation. As [Morrison 1986] noted typically the customer actually wants to know one of two things:

1. Will my new system design "work"?

2. Why doesn't the existing system work?

Both of these questions presume that the analyst knows about the type of system he is attempting to analyze. Therefore, the process of formulation of analysis requirements must include the development by the analyst of a perception of the system, formulation of notions of symptoms and concerns, identification of possible problem classes, and conditions for the solution of these problems.

Unlike the system designer, however, the simulation analyst need not formulate the complete necessary conditions for the solution to a problem, in fact generally what he must produce is the inverse (i.e. the conditions under/which the problem would be identified). These conditions then simply become the requirements for the model which will be designed. The model and experiment must clearly indicate a problem situation (if one exists). This implies that the analyst must at least implicitly anticipate the problems which could exist in a system. Hence the benefit of experience in analyzing a particular type of system!

The preparation of the analysis requirements then includes the assimilation of both the kind of information which a designer would need to design solutions [Mesarovic et al. 1970], and the kind of information a diagnostician would need to diagnose a problem. Both cognitive skills require knowledge of the system architecture, the system elements, their properties, and the connectivity (i.e. how one element affects another) which is essentially the theory of system dynamics. Fortunately for the analyst, models exhibit many of the same features as less formal natural languages (as will be described in some detail in Section 5, and 7). One of these characteristics is the *efficiency* of models (i.e. the capability for the same model to mean different things in different situations). In fact this capability is what actually distinguishes a model from a specification. This feature allows the analyst to extensively use default assumptions concerning the conditions which the analysis system must satisfy. The process of model design, construction, and experiment execution, then, can be used to some extent to debug these specifications.

2.8 Analysis and Experiment Planning

The planning activity of the simulation analyst is one of the most complex and central issues to the conduction of a successful simulation analysis. It is during this planning activity that the analyst merges the analysis requirements with his knowledge of such constraints as:

1. The resource constraints of the project,
2. The constraints of mathematical simulation methods,
3. The constraints of statistical methods,

4. The data availability constraints,
5. The programming language constraints of a particular simulation model implementation language.

The experienced simulation analyst brings the knowledge of these constraints to bear on the development of an approach which includes:

1. An overall time sequenced plan for the analysis activities (i.e. how much time (and effort) will be spent in the gathering of system descriptions, design of the model and experiments, collection of data, analysis of data, model construction and verification, model validation, experimentation, and results analysis),
2. The level of abstraction to be used, and a preliminary model architecture,
3. The associations which will be made between the model components and the system description objects (i.e. the modellers view of the real world),
4. The property specifications which will be required to implement the causal behavior required and to gather the data for analysis,
5. The preliminary experimentation strategy,
6. Estimates on the actual time and resources required to conduct the analysis,
7. Strategies for display and explanation of the analysis results.

The planning of an experiment to be run with the model is also driven by the goal or purpose of the analysis. There can be a number of different reasons for doing the analysis including: evaluation, comparison, prediction, sensitivity analysis, optimization, establishing functional relations, studying transient behavior, finding bottlenecks, etc.

The analyst uses the purpose of the study to determine the statistical objectives to be achieved by the experiment. This sets the line of interactive questioning that would be followed with the customer to define the experiment. One of the difficulties encountered in this process is the fact that a particular study might have more than one "measure of effectiveness". In such cases it will be necessary for the customer to designate one as the primary and the others as secondary. More commonly, the customer is not aware of any particular desire for any particular "measure of effectiveness". In fact the "measures of effectiveness" are a side effect of the analysis requirements as established by the analyst. In addition to "measures of effectiveness" the analyst must also concern herself with the identification of measures of "indication", "approximation", and "imitation" as will be discussed in Section 7. These measures are used by the analyst to calculate required sample sizes, etc., and to identify the mechanisms which will need to be in the model design to allow collection of specific data.

As stated earlier, the goal selected will determine the line of interactive inquiry that the analyst pursues. For example, assume the customer says the goal is to explore functional relationships between system elements . This implies that a factorial design will be required. The analyst would then solicit information as to what variables are of interest, the highest and lowest values of the range of interest, whether each variable is quantitative or qualitative etc. From this information (as well as the measure of system effectiveness to be used), the analyst can determine a factorial design to be used as well as the number of replications to be run. The customer would then be solicited for other statistics that might be of secondary interest. From the above dialog, the analyst designs the experiment to be run to obtain the answers needed.

Thus the analysis and experiment planning activities encompass the development of a high level design of each of the components in the analysis activity as well as the delineation of the tasks involved in these activities. The mechanisms which are employed in this more structured planning activity have been addressed by many of the AI studies of planning. The state of the art in understanding planning processes is best understood by reviewing the historical developments in AI directed towards duplication of these capabilities. AI efforts in planning have generally focused on the following problem areas:

1. General Methods (e.g. GPS),

2. Treatment Planning (e.g. Emycin),

3. Robot Task Planning (e.g. STRIPS, ABSTRIPS, HACKER),

4. Laboratory Task Planning using constraint propagation (e.g. MOLGEN),
5. Story Planning/Text Understanding (e.g. PAM, PEARL, PANDORA, FAUSTUS, ELI),
6. Assembly Instruction Planning (e.g. NOAH),
7. Repair Planning (e.g. NONLIN).

From a methodology point of view, approaches to planning can be separated into four major classes [Cohen and Fiegenbaum 1981]:

1. Non-hierarchical planning,
2. Hierarchical planning,
3. Script based (e.g. variant) planning,
4. Opportunistic planning.

2.8.1 Non-hierarchical Planning Systems

Non-hierarchical planning systems work with a single representation of the plan, developing a sequence of tasks to achieve the goal of the plan. While the plan itself may have a natural hierarchy, the system produces this plan in a linear sequence of steps. A complete plan is not available until the last step (at whatever level of detail) is determined. Thus the systems do not have the capability to distinguish which tasks are key to solving the problem versus those tasks which are simply there for resolving details. The fact that these systems often are capable of generating and manipulating complex hierarchies of goals and subgoals often makes the identification of an appropriate classification difficult. The advantage of non-hierarchical planning approaches is in the simplicity of the planning system structure. The STRIPS [Nilsson 1980], HACKER [Oshea 1984], INTERPLAN [Cohen and Fiegenbaum 1981] systems are examples of non-hierarchical planning approaches.

STRIPS was the "original" planner that explored the concept of associating add-lists and delete-lists with an action as a means of representing rules. Means-ends analysis provided the basis concept for the processing cycle that was used in STRIPS as well as in such later planners as MOLGEN. The cycle may be summarized as follows:

1. Compare the current goal to an initial state.

2. Find differences between the two.

3. Look for an action to reduce the difference.

Besides the problem of failing to distinguish between major considerations and minor details, STRIPS suffered from two other notable limitations:

1. Since goals that could not be immediately satisfied were simply stacked in a LIFO manner, subgoals could be satisfied repeatedly in the course of solving a problem.

2. Since the comparison step in the planning cycle could produce multiple differences, and since alternative actions could be available to reduce a given difference, many paths to a goal could exist. Yet no provision was made in STRIPS for choosing wisely among these alternatives.

HACKER addressed the general problem of how to deal with conjunctive subgoals by including a capability to reorder subgoals and try again if an initial arbitrary ordering produced a dead end. Philosophically, the HACKER design made the assumption that people generally plan by making a hastily contrived plan and then debugging it. That is, any ordering of steps is assumed to be as likely to be correct as any other. Later systems, such as NOAH and MOLGEN used the alternative least commitment approach; but it should be noted that the latter approach assumes a sparse solution space, contrary to the HACKER assumption. In the "analysis planning" domain the experiment planning subtask appears most amenable to the HACKER approach.

INTERPLAN varied the HACKER approach to ordering goals by adding the capability to promote subgoals over superordinate goals if the original ordering proved futile. This approach may result in achieving a goal more than once, but in making possible a solution that HACKER could not find.

The Waldinger system further varied the general HACKER approach by adding the concept of goal regression. Basically, Waldinger noted that a goal that has already been achieved can by violated by a subsequent action, but that the goals can be reordered such that the offending action and its goal are moved back out of the way of the goal that would have been violated otherwise.

Non-hierarchical planning approaches do not appear to approximate any of the analysis planning tasks presented above. What they do provide is some insight into the basic mechanisms for plan construction which are used in the hierarchical, variant, and opportunistic methods described below.

2.8.2 Hierarchical Planning Systems

Hierarchical planning systems on the other hand create and manipulate complete plans at various levels of abstraction. The general idea is to be able to completely reason a solution to lower levels of detail. Different approaches to hierarchical planning are distinguished by the choice of what is abstracted. For example, ABSTRIPS [Nilsson 1980] abstracts the goals of prioritization of the importance of the classes of goals both in the statement of the problem to be solved and in the description of the rule conditions and effects. Thus ABSTRIPS produces a complete plan for the "important" part of the problem using tasks which have been simplified to consider only the important constraints and produce the important results. Once this high level plan has been worked out, the ABSTRIPS system attempts to satisfy the next level of goals within the framework of the high level plan. The basic idea is to split the logs before worrying about shaving the bark! The set of operators can be essentially the same at each level; what changes are the conditions under which an operator can execute and the effect that an operator (task) has.

The NOAH system, on the other hand is designed for those planning problems in which a hierarchy of tasks naturally exists. For example, building a wall breaks down into activities of setting a foundation, procuring bricks, mixing mortar, laying the bricks, and tacking the corners. Thus the NOAH system attempts to create levels of plans with the appropriate level of problem solving tasks. When the interactions of the tasks have been resolved at a high level NOAH then attempts to refine the plan to the next lower level by refining the individual operators, and reassessing the interaction constraints at the lower level. Thus NOAH type systems can be said to abstract the problem solving operators.

The third type of hierarchical planning system abstracts both the operators (tasks) and the objects which those operators create or use. The MOLGEN [Stefik 1981] system is an example of this class of system. Thus the brick and mortar example in a MOLGEN framework would be planned at the build wall level, the procure materials and tools level, and then the detailed activity level. The MOLGEN planner might generate specific constraints that the bricks must satisfy in order for the detailed activity to be successfully accomplished. MOLGEN effects the refinement of tasks and objects through a process of constraint propagation, and delayed binding. Simply, constraint propagation means that the planner has the capability to move horizontally or vertically through the levels of plans depositing additional constraints which the activities must accomplish or the objects must satisfy. Delayed binding refers to a least commitment strategy for decision making at any level. This strategy (which is similar to "Just in Time" decision making in system development methodologies) means that the system attempts to be as noncommittal as possible in selecting a specific task or object. The general idea of such a strategy is that by delaying the decision making more information will be available to make the correct decision thus reducing the number of decisions which must be undone and reducing the backtracking. This type of mechanism appears to be consistent with the activities in model abstraction and preliminary experiment planning identified above.

Besides the hierarchy introduced in the planning task itself, MOLGEN also implements a hierarchy in its own control structure. That is, it plans about planning using a three level structure representing strategy, design, and domain knowledge. In separating the strategy and design levels from the domain level, it reaches for a means of planning over multiple domains. Relative to the characterization of the "analysis planning" activities of the simulation analyst this type of planning provides an excellent characterization of the model architecture planning. That is, the analyst treats the development of the preliminary model architecture as the planning of a set of mechanisms which will produce the desired data goals. The high level architecture is viewed as requiring a series of subtasks to be realized. These subtasks are cast as model components and the process continues in a recursive fashion.

A more recent system, SIPE [Wilkens 1984], extends the NOAH concepts of hierarchical planning and parallel actions, borrows somewhat from the MOLGEN idea of propagating constraints on objects, and adds a few ideas of its own. Designed as an attempt to achieve domain independence in a planning system, SIPE has generated plans in four domains: a blocks world, cooking, aircraft operations, and travel planning. One of the major contributions of SIPE is the introduction of a formalism for describing operations that explicitly ties both objects that participate in an operation and constraints on those objects to the operation itself. It also introduces the notion of resources needed by an operation, though there is no concept of a resource type or of a decision making basis for resource allocation. The operation formalism serves further to clarify the relationship between various levels in the hierarchy. That is, preconditions (which would have formed subgoals in earlier systems) are used in SIPE to determine the applicability of the operator and to identify explicitly conditions that should have been satisfied at a higher level in the hierarchy. Conversely, goals in SIPE specify those subgoals that must be achieved at the next lower level in order to accomplish the operation at the current level. Thus, the goal portion of the operation record tells how the operations can be done while the precondition portion of the record tells what must have already been done.

A second contribution of SIPE is an extension of the NOAH method of dealing with ordering constraints on operations. NOAH could identify harmful interactions through the TOME; SIPE is able to identify also interactions that are fortuitously helpful to the planning process. A third contribution is the introduction of deductive operators that serve to determine consequences of actions beyond those explicitly mentioned in add-lists / delete-lists. A fourth contribution concerns the scoping of actions. When a node was expanded in NOAH it was assumed that the last node generated in the expansion was the one that actually accomplished the higher level action. In SIPE the parent node can specify which node in the expansion serves this purpose. Further, nodes in an expansion can specify which other node in the same expansion is supported by their action — thus establishing a scope for a node. This helps in identifying the purpose of the node as well as in avoiding harmful interactions. SIPE is also the first system to establish a classification for constraints, effectively setting a basis for a generalized constraint language. Initially designed as a highly interactive system, in part to allow human decision making to aid the system during the development process, SIPE is still under development.

2.8.3 Script or Variant Planning Systems

Script or variant planning approaches in the AI field are very similar to the variant approaches attempted in process planning. In fact the HICLASS system [Liu 1987] is an example of an AI based generative process planning system which is based on an AI script approach. In the script worlds the planning system has a data base of skeleton plans which are pattern matched against a problem situation. The "best fit" plan is then refined to accommodate the problem specific details. This type of planning strategy has been used extensively in text and story understanding and generation (hence the origin for the term script) [Schank and Abelson 1977; Winograd 1983; Wilensky 1983]. While this type of planning may appear too simplistic a model for many of the analyst planning functions it is a reasonable model of the analysis project planning activity where the gross level tasks tend to follow a predefined sequence.

2.8.4 Opportunistic Planning Systems

Opportunistic planning is based on a paradigm of human reasoning [Hayes-Roth and Hayes-Roth 1978] whose essence is that the human planner does not generally develop a plan all in one piece, either at the detail level or the abstract level. Rather the planner tends to develop plans for parts or clusters of the problem. These clusters may be planned using the planning strategy which is appropriate for that particular part of the problem. The clusters are linked together to form larger portions of the plan as the opportunity arises. Implementation of the Hayes-Roth model can be viewed as a set of parallel processes communicating through a common database containing the current state of the planning process. The cooperating specialists address different parts of the problem as the information which they need is posted on the "blackboard".

The Hayes-Roth model recognizes five separate planes of planning decisions. The first plane is called the "plan" plane. This plane is the plane of operations planning at various levels of abstraction. The second plane is called the "plan abstraction" plane. Decisions in this plane motivate the initiation of specialists in the first plane. The third plane is called the "meta-plan" plane. The decisions made at this level are about how to solve a particular part of the problem. Thus issues concerning the choice of a problem solving model or strategy are resolved at this level. An interesting and important concept which is addressed at this level is the formulation of the concept of what constitutes a "well formed" plan. The fourth plane is called the world knowledge plane, which contains the various levels of world knowledge available to the system. This includes knowledge about the problem situation, and constraint knowledge which is not specific to the rule structures of any one specialist. The fifth plane is called the executive plane. The decisions with respect to resource allocation, scheduling, and access to information are made at this level.

This "opportunistic" planning model would seem to best characterize the *approach* that the analyst takes towards the overall analysis task. That is, he may work on the model architecture design for a while, then switch to the experiment planning, then when a problem is detected, or when there is simply not enough determined about the model to continue with the experiment planning, he will revert back to the model planning. One of the powerful features of the "opportunistic" model is that it places few restrictions on the planning models which are used for the subparts.

2.9 Model Design and Specification

The design of a model is similar in many respects to the design processes which we have already discussed. The analyst has in mind the analysis goals, the system description, and general constraints of the implementation mechanism at his disposal. He also is aware of the resources available for the analysis in terms of personnel time, computer resources, etc. The design approach depends considerably on the level of detail of the system description, and on the analysis goals. The design process itself depends on the component base and the experience of the designer in using these components. Of the design models presented earlier in this section, the external constraint approach appears to best characterize the process most often used. This is particularly true in entity tracing types of simulations. Through the construction of a scenario of processing of an entity or series of entities the modeler identifies (or establishes) the requirements for model components which perform:

1. Creation of an entity,
2. Assignment of attribute values,
3. Monitoring of interesting statistics,
4. Resource consumption,
5. Time delays,
6. Flow (routing) logic,
7. Processing logic.

The model design is generally performed in the context of an existing model plan conceived during the analysis planning phase. The analysis plan is used to focus the scenario construction process.

Model specification refers to the detailed design of the model components which were previously identified in the model architecture design process, as well as the specification for the interfaces between these model elements. With existing simulation languages this detailed specification can often times take the form of the coding of the model in a third generation language (e.g. Slam, Siman, GPSS, Autosimulation, etc.). The major difference conceptually between the model design and the model specification is that in the specification stage the analyst is much more concerned with the behavior of the modeling engine itself, and the implementation details / constraints of the specific programming language. Thus in the model specification phase such issues as the precise specification of the representation of the values of the attributes of the model objects need to be considered. Issues relative to parameterization of the model to support the execution of the desired configurations for the runs of the models required by the experimental design must be determined. Similarly the programming details associated with the implementation in a particular machine environment (sizes of data versus program, overlays, execution costs, etc.). Finally the programming of the results analysis routines, interface to data files, interface to statistical analysis packages and graphical animations must be addressed.

Addressing the full scope and characterization of what a programmer knows about programming is beyond the scope of this research (for insight into this area see [Minsky 1974; Soloway et al. 1982; Waters 1984]). What is important to note is that these programming activities are actually a small part of the overall model design and decision making process and that current (or near current) simulation support environments already provide many good support tools for these activities. What we will attempt to provide is some insight into the knowledge structure, sources and reasoning mechanisms which a modeler uses to construct his abstractions of the world which enable him to formulate the model constructs which are the focus of the specifications. In Section 5 we will present a formalization of situation recognition which includes a formalization of action, time and change which we will use in Section 7 to construct a mechanism for implementation model generation.

2.10 Results Interpretation

Results interpretation is what the analyst does with the results of the analysis after the experiment is conducted and the data has been statistically analyzed. The primary activity of this task is the construction of a "theory of how things work" which can explain the output statistics of the simulation analysis. Another equally important activity is the generation of "why" the current customer's "theory of system dynamics" does or does not predict the observed behavior. Sometimes this is as simple as showing a void in the considerations taken into account by the customers understanding of his system or design. At other times it is a process of identifying errors in the "logic" which the customer is employing. Most often however, it requires the education of the customer into a new "theory of system dynamics" which can require considerable explanation and example generation. For example, if the customer is laboring under the belief that idle machines mean lost production then explaining the fact that within tightly coupled systems scheduling machines to maximum capacity can be disastrous will be difficult.

One of the most important aspects of "results interpretation" is the translation by the analyst of what he has learned from his model of the world into general "truths" which the customer can use. That is, he must be able to generalize the results he has obtained into axioms which the customer can utilize in his understanding of how the system works. The customer can then reconcile these newly discovered constraints with those he already holds. This reconciliation can result (and often does) in the identification of inconsistencies. If the customer resolves these inconsistencies in favor of the analysis results then he modifies his theory of the system dynamics. Otherwise, he rejects the model (or at least the interpretation of the analysis results).

Part of the difficulty encountered in results interpretation is sourced in the type of question the analyst is attempting to answer. Questions which involve "possible worlds" types of reasoning (e.g. *Can the west end loader system handle 80 jobs per hour?*) require the analyst to explore configuration or operational rule structures which are completely foreign to the experience base of the customer. Should such a "possible world" prove to exhibit the desired performance behavior the analyst is faced with both the task of communicating the new system description as well as the explanation (in terms of operational axioms) which justify his results.

Results interpretation can thus be couched in terms of the system description process described in Section 2.2.1. That is, the analyst must describe the predictions of the experimentation in terms of a scenario with the important cause / effect relations explained. Such an activity goes beyond presentation techniques such as animation or graphical summarization of statistics. It requires an in-depth knowledge of the preconceived notions of the customer, the relation of the model structures to this understanding, and an explanation strategy. The presentation techniques are merely mechanisms in the explanation strategy. The underlying goal is to explain how behavior comes about. It is our conclusion that this process of results explanation must rely heavily on arguments based on qualitative reasoning. In this light it will be proposed in Sections 3 and 7 that qualitative simulation methods be used not only as a basis for quantitative simulation model design and generation but also as a basis for automation of the results interpretation process itself.

2.11 Summary

The goal of this section was to provide a characterization of the cognitive processes involved in decision making based on the use of models and simulation. This characterization was initially presented from a high level view. Subsequently each step in the process was examined and models for the reasoning processes in that step were proposed. One of the key points made in this chapter is the recognition that simulation models are not descriptions but designed abstractions from descriptions. Another key point that was made is that the customer uses internalized theories of dynamics to qualitatively evaluate proposed causes of problems and proposed solutions to these causes. These two points will show up as driving requirements for the KAMSS architecture described in the next section. They also provide much of the structure and motivation behind the semantic theory and reasoning strategies presented in Sections 5 and 6.

3. KAMSS ARCHITECTURE

This section outlines a design for a knowledge based support environment for decision making using modeling targeted to simulation analysis. This architecture is based on a number of unique concepts which differentiate it from current simulation support environments. These concepts include:

1. Support for the system description process separated from the system modeling process.
2. Use of a system description representation based on a generalized semantic model rather than a mathematical model-theoretic semantics.
3. Generative model design as well as a library of reusable model types.
4. Provision of both sketch and natural language input in a "dialog" mode.
5. Full access to the underlying knowledge representation mechanism in all modes of the user interface.
6. Support for qualitative analysis of a system description as a form of common sense reasoning about system dynamics.
7. Support for object and rule based simulation analysis.
8. Use of annotated text as a mechanism for the implementation of long term storage of a knowledge base.
9. Support for the interpretation of simulation experiment results in addition to the more traditional support for statistical analysis of simulation results.

This architecture is also unique from the point of view of the interface style that it provides. We will refer to this style as "common-intuitive". A "common-intuitive" style of user interface to a computer software system is one which attempts to accurately reproduce the *situation of an activity* were that activity to be performed without the computer. Many of the "visi-" microcomputer-based systems attempt (sometimes rather successfully) to provide this style of interface. In the KAMSS there are really two role models which should be supported. When the KAMSS is in use by the customer, he would expect it to respond and act in ways similar to the way a human analyst responds. When the system is in use by a systems analyst, he would expect that the system would act like a customer in some modes (when answering questions about a system) and like a programmer, statistician, or designer in other modes.

The previous section provided a characterization of the process of decision making involving the use of modeling and simulation. What is important is that such a process involves a large number of complex cognitive skills. A system to support such a collection of activities will involve the application of either a large number of specialized skills or the use of a generally weak reasoning process with a large knowledge base. We have chosen the latter approach in order to provide a generalized platform which can be customized during use. The design of an effective environment for automated support of that process must provide the capability of either transparently isolating a large number of those portions of the task which it cannot support, or providing a general set of mechanisms which can be extended during use. The approach of this research is along the lines of the second alternative.

The KAMSS concept calls for an extension of the notion of user profiling to include the concept of usage situation. The usage situation accounts for the perceptual limitations of the KAMSS as an imitation of the human performing in a similar situation. Thus, limitations in perceptions caused by the fact that the mechanisms only allow at most for keyboard and mouse input must be taken into account in the design of the knowledge systems responsible for the discourse management. The prototype components of KAMSS constructed to date have made extensive use of presentation objects in presentation style interfaces to effect this type of user interface action anticipation [Aiello et al. 1981; Genesereth and Ginsberg 1985; Symbolics 1986]. The usage situation takes into account not only mode of use but the existence of description or model fragments as well as the semantics of the ontology underlying the input.

Figure 3.1 provides a logical view of the KAMSS. This view shows three primary modes of operation. The normal usage of these modes would depend on the type of user. The manager, for example, would start with the creation of a system description (via the SDCE), move to the design of a model (via the MDS), and then possibly request a particular scenario to be simulated (invocation of

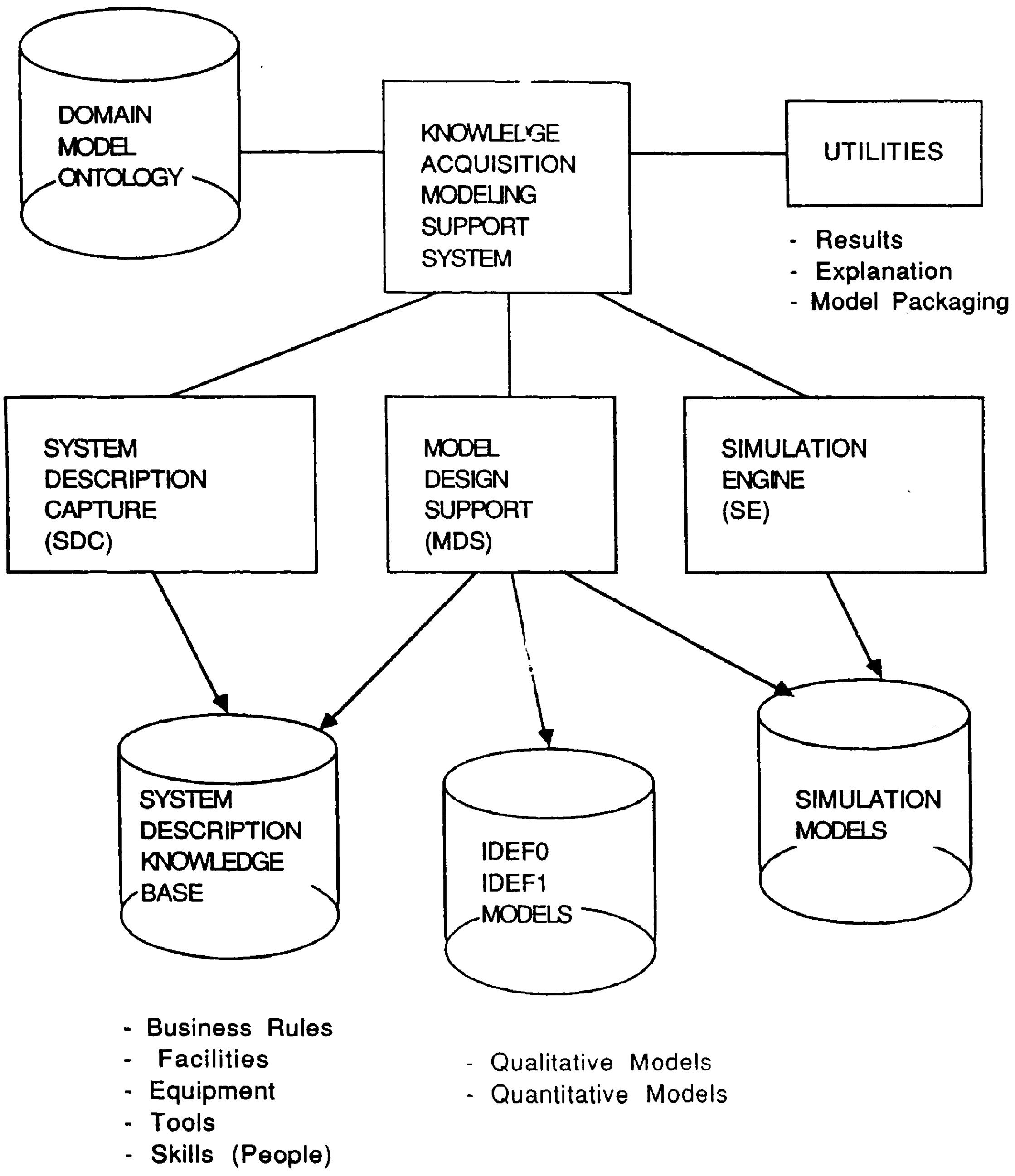


FIGURE 3.1: LOGICAL VIEW OF THE MAJOR COMPONENTS OF KAMSS.

the SE). A simulation analyst, working on the design of a model which was too complex for the MDS generation capability would enter the MDS and use its interface to the System Description Data Base (SDDB) to obtain a system overview, and then refine the model architecture generated by the MDS (and stored in the Logical Model Library) for the particular problem. The analyst can also directly access, edit, or create "Implementation Models" stored in the Implementation Model Data Base (IMDB). The differences between "logical models" (LM) and "implementation models" (IM) will be discussed in more detail later in this section. Essentially the term logical models cover models constructed / generated for the purpose of supporting qualitative simulation and "common sense" reasoning. Implementation models cover traditional quantitative simulation models and stand alone or integrated application programs.

3.1 Design Rationale

Decision making using simulation requires a great deal of expertise to translate the needs, goals, and objectives of the user into the appropriate model, using the right data, executed under the correct design of experiment and analyzed appropriately. The use of KAMSS will follow a very different strategy in which the manufacturing decision maker defines knowledge about the system (i.e., the description of the objects in his environment and the business rules of operation, etc.) through a dialog with KAMSS. The KAMSS system would participate in the evaluation of symptoms and concerns which the decision maker identifies. This "capture" of a system description is supported by an extendible type and class based ontology which generates a framework that is tailored to the "ontology" represented in the type / class network. This framework provides both form and structured text input of system descriptions organized around visual layouts (block diagrams or sketches).

In posing questions to the KAMSS, the user invokes data retrieval and deduction based on the current state of the system descriptions. Inability to answer the question directly from the knowledge base triggers the KAMSS to attempt to deduce the needed results using theorem proving methods described in Section 6. As the KAMSS maintains both class and type based representations in its knowledge base, these methods could cause the dispatching of remote query processes across the network interfaces to the manufacturing / engineering or business systems to acquire the data necessary to respond to the query. Failing those methods, the second course of action is to generate a "token" or model world in which constraint propagation (primarily over temporal and mechanism constraints) is used in conjunction with qualitative simulation to derive the needed answer as described in Section 7. Failing these methods KAMSS would use the results to pose a list of tests, measurements, or observations which the user would need to perform. Through this interaction, the need for math model based simulation analysis may be identified and, in that case, the KAMSS would design the appropriate model. Extension to the background knowledge base of such a support environment to include experience rules for generation of design alternatives in a particular domain would allow the manufacturing user the option to enter design goals and let the system generate the series of models and model runs necessary to find an acceptable design solution.

The KAMSS design must include the foundation software packages and a system architecture that will be easily adaptable to end-user requirements in a broad range of business environments and system development applications. The adaptability should allow for both expansion and contraction of the system to allow for tailoring to various levels of hardware capabilities. A realistic design goal should be speeding up the system modeling and/or simulation process by a factor of ten to a hundred over current methods. The design must encompass a system description capture environment as well as a model generation and simulation support environment. It must also provide an engineer's programmer's workbench, that is, a programming environment that provides the "occasional" programmer with support for; generation of analysis utilities to package around a standard analysis program, generation of stand alone analysis tools driven off of data acquired through the SDCE or a MDS session, or rapid prototyping of manufacturing engineering knowledge based applications.

Necessary features of the KAMSS include:

1. Interactive natural language and sketch user input,
2. Full access to the description ontology base for the user interface support utilities,

3. Intelligent assistance to the system definition process including:

    3.1. End-user extendible classification ontology,

    3.2. Data acquisition framework, form and structured text input generated from the classification ontology,

    3.3. Classification support to assist in the instance typing process,

    3.4. Support for hypertext use throughout the classification structure and description knowledge base,

    3.5. Provisions for location, copy / merge and edit of element descriptions.

4. Intelligent support of the model design process,

5. A reasoning mechanism for performing qualitative problem analysis and business rule simulation,

6. An object-oriented, rule-based simulation engine for math modeling based simulation,

7. Dynamic output display using animation graphics,

8. Automated statistical analysis programs,

9. Intelligent support for the interpretation of the simulation analysis results relative to the model design and analysis goals,

10. Interface construction utilities for integration with factory information systems,

11. Interface utilities for piping results from one analysis instance to another to form an "analysis scenario",

12. Utilities to support the packaging of decision scenarios for use outside the KAMSS environment.

The primary goal is to support management decision making by automating the analyst role, in the modeling and simulation process, using a knowledge based support environment that can perform many of the functions which traditionally are performed by the human analyst. For this to happen, it is necessary to step back from the mire of traditional simulation languages and develop a modeling and simulation system which provides a more natural environment for system description, model expression, and experimentation. There are important differences between what is being done today and the proposed knowledge based modeling and simulation system. The primary one is the desire to move away from the traditional focus on the modeling of an environment to focus on a system description which can be used as the basis for generating models of many different varieties.

In order to use simulation correctly and intelligently today the practitioner is required to have expertise in a number of different fields. This generally means extensive knowledge of probability, statistics, design of experiments, modeling, computer programming, and simulation languages. This translates to about 720 hours of formal classroom instruction plus another 1440 hours of outside study (more than 1 man-year of effort) — and that is only to gain the basic tools. In order to really become proficient, the practitioner must then go out and gain real world, practical experience (hopefully under the tutelage of an expert) [Shannon and Mayer 1986]. The goal for the development of the KAMSS is to make it possible for engineers, scientists, and managers to do system analysis and simulation studies correctly and easily without such elaborate training.

Regular demands for simulation analysis of ever increasing complexity is becoming the norm in many environments. For example, simulation support in a large corporation requires capabilities to model at several levels. Modeling at the manufacturing engineering and manufacturing management level for each operation is required both to plan new facilities and to investigate problems with existing facilities. Modeling at the vice-president of manufacturing staff level must support the integration of multiple models of upwards of 20 operations in order to evaluate the product and manufacturing strategic plans. What is needed is a common definition of each of the production operations and of the company as a whole, from which different models from different viewpoints could be constructed. Such a system definition would be carried forward year after year, in contrast with the individual non-reusable models currently being developed. One of the design goals of KAMSS is to establish the technology to allow the decision makers and their staffs to quickly generate models needed for a decision scenario with minimum support from experienced modellers. Such support can only be provided if a rich description base has been put in place. However, the description base can be collected by the domain experts over a longer period of time, using the SDCE component of KAMSS. This will free the systems analyst and simulation modeller to focus on creating / expanding the modeling knowledge of the system itself. This goal requires natural language and sketch input capabilities for the creation of the system descriptions and intelligent support of the model generation based on that system description.

3.2 Usage Scenarios

As described in the section covering system design in Section 2, one of the most common ways to explore the requirements for a new system is to try to delineate scenarios of use of that system. In this section we summarize the customer and analyst tasks which the KAMSS is intended to support and then define nine major scenarios of use of the KAMSS. The intended usage scenarios of KAMSS described below are modeled after the interaction process described in Section 2. Through these scenarios we have characterized the philosophy of KAMSS operation and scoped the requirements on the architectural approach to the KAMSS design.

Customer / User Tasks Supported:

It is envisioned that the customer would use KAMSS to support the following tasks:

1. Ongoing development of an evolving description of the facility,
2. Rationalization of an observation as a symptom or concern,
3. Situation explanation,
4. Problem analysis,
5. System change planning,

6. Model generation,
7. Simulation modeling,
8. Causal reasoning and qualitative simulation,
9. Analysis application generation.

Thus, from the customer / user point of view the KAMSS should embody the capabilities, knowledge, and characteristics of an experienced systems analyst with simulation modeling capabilities.

Analyst / User Tasks Supported:

The KAMSS philosophy recognizes that initially the analyst will be a major user of the KAMSS system. Even with the capabilities proposed for direct customer use, it is probable that the initial users will still be systems analysts who have been charged by the customer to assist them in the solution to some problem. It is also the case that in many instances the customer is acquiring analyst support because of the system design expertise of the analyst as well as the modeling expertise. It is envisioned that the analyst would use the KAMSS to support the following tasks:

1. Organization of acquired system description information and data,
2. Access to the current system definition,
3. Causal reasoning and qualitative simulation,
4. Access to previously defined models for the purpose of re-execution,
5. Model design support,
6. Experiment design support,
7. Support for the interpretation of experiment results relative to the current system configuration,
8. Configuration of "deliverable" or packaged versions of a decision scenario,

9. Construction of specialized factory information system interfaces to a decision scenario,

10. Application development.

It is anticipated that the analyst / user would require support for many of the same tasks as the customer / user. In fact, the best way to characterize the analyst / user is as an "experienced" customer / user.

3.2.1 Scenario #1 Acquaintance

The initial encounter of the user (customer or analyst) with the system will be the most difficult to manage from both the user's point of view and the KAMSS point of view. The goal of the system design is to require minimal training for use. Thus, the user would expect to log in to the system and begin a dialog in much the same structure and format as a dialog with a human analyst. Part of this dialog will be the explanation by the KAMSS of its role and the services it can perform. During these initial sessions the KAMSS would be expected to develop a user profile of the customer. This includes the construction of a user specific lexicon, and a model of the tasks which the user performs, as well as the initial model of the user's perception of his environment. As will be described in a later section, the KAMSS has a system description (similar to a conceptual schema) which "knows about" typical usage patterns. This usage profile allows the system to "anticipate" the functionality requirements of the typical user. The user profile is merged on top of the typical profile so that it is always available as a default profile.

3.2.2 Scenario #2 Capture of System Descriptions

A high-level view of the overall scenario of a system description capture process is illustrated in Figure 3.2. This scenario can be viewed as having three major subparts. The set-up mode involves determination of the adequacy of the existing knowledge representation structures for capturing the description of the situation at hand. If inadequacies are detected then the underlying framework must be extended incrementally. The acquisition mode covers the actual collection and typing of the system description. The usage mode in this scenario refers primarily to the validation process via browsing of the assembled description, invoking of summary generation, and specific query processing.

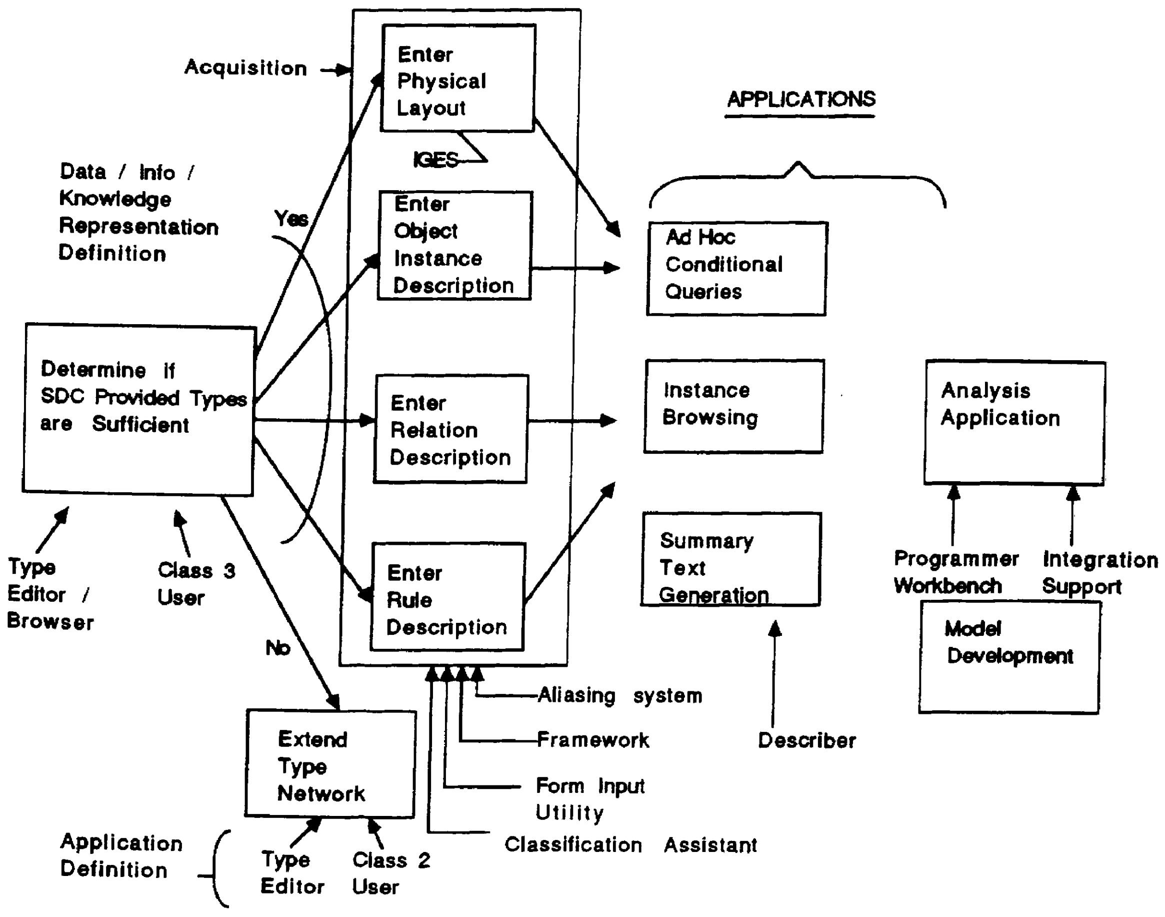


FIGURE 3.2: SYSTEM DESCRIPTION CAPTURE SCENARIO.

The term "textual input" is slightly misleading since the actual form of input is intended as an interactive dialog between the user and the system. This dialog will include the use of:

1. Forms and menus,
2. Sketches and structured diagram input (see IDEF1/ES in Section 8),
3. Commands,
4. English declarative and interrogative sentences,
5. Context sensitive notes,
6. Files of preprocessed text descriptions,
7. KAMSS generated queries,
8. KAMSS generated summary text, and figures.

The use of a dialog style of interaction allows the system the advantage of being able to disambiguate user text via formulation of direct questions. The difficulty arises from the fact that between sessions with the user the system must be able to "remember" the context of the previous sessions. Part of that "memory" is encapsulated in the system description which is constructed from the previous session, as well as the user profile and sketch inputs. However, there are other more transient memory elements which must be managed such as:

1. Unanswered questions which may require repeating portions of what "was stated" in order to jog the memory of the user.
2. Portions of contextual structures that were built for the process of "understanding" the user's statements.
3. An "agenda" of items which the user has mentioned but not yet completed the description for.

A typical interaction scenario is illustrated in Figure 3.3. An advantage of the interactive dialog form of input is that the system can restrict the user to simpler

```
input -> OUR FACILITY PRODUCES SHEET METAL PARTS FOR
           AIRCRAFT APPLICATIONS.

input -> PRODUCTION ORDERS ARRIVE FROM PRODUCTION
          CONTROL DAILY.

input -> THE TYPICAL ROUTE FOR AN ORDER IS; SHEAR, FORM,
          PUNCH, DEGREASE, PAINT, LABEL.

input -> O.K?

question --> is an order for a single part?


input -> YES


question --> What is an order ?


input -> ORDER IS A JOB RANGING IN SIZE FROM 1 TO 2,000
         PRODUCT UNITS.
```

FIGURE 3.3: TYPICAL TEXT INTERACTION WITH KAMSS.

syntactic forms of input. After all, one of the favorite interview ploys used by human analysts is the "play dumb" mode. In this mode the analyst conditions the customer to use simple direct statements by simply questioning every complex statement uttered.

One of the primary issues which must be addressed in the acquisition portion of the KAMSS is proof of competence. That is, how will the user "know" how much KAMSS has "understood" of what it has been told. We believe that the best way for the illustration of the interned knowledge of a particular manufacturing situation to be displayed is through the generation of summary textual descriptions. The ability to respond to queries about the description provided would also be an effective mechanism for the illustration of the "understanding" which KAMSS has of a particular situation. However, the reasoning required to answer particular classes of questions begins to get into the analysis scenario domains and so will be discussed in a later section. An example of the summarization which would be produced after interning of the text illustrated in Figure 3.4 is displayed in Figure 3.5.

Another issue associated with input of the system description is the need to be able to extend the capability of KAMSS to "understand" the input presented. This requires the capability to interactively (and incrementally) extend the ontology portion of the KAMSS knowledge base as well as the system description itself. This can be accomplished (as in the prototype SDCE) by integration of the knowledge representation structure into all of the user interface facilities. A change in the ontology structure must automatically generate the necessary form and declarative statement structures necessary for acquisition of instance data understood by the additional concepts. It also requires the ability to freely associate both concept types (or classes) and instances. Finally, it becomes necessary to allow annotations about any concept or instance. This last feature is provided in the SDCE prototype through a "hyper-text" capability. Such an approach has the additional side effect that the representation system can be used as a sophisticated application program generation utility.

Examples: Process Flow for Cab Elevator - BIW to uniprime.

1. Cab stopped and located.

2. Box (or empty box carrier) stopped behind cab. Read by limit switch to see if box is present.

3. Elevator/transfer extends beneath cab - 6 sec.

4. Elevator/transfer lifts cab off carrier - 2 sec.

5. Elevator/transfer retracts with cab - 6 sec. After this step, the cab carrier is now empty and is released. If the path is clear, the cab carrier proceeds through the box transfer station (on BIW conveyor) back to the body shop. This is simultaneous with - the matching box (or empty carrier) behind the cab is released to the box transfer station where the box is loaded (if present) or continues through if no box is present.

6. Elevator/transfer lowers with cab - 10 sec.

7. Elevator/transfer extends with cab - 6 sec..

8. Elevator/transfer lowers cab to position on uniprime carrier 2 sec.

9. Elevator/transfer retracts from beneath cab - 6 sec. After this step, the uniprime carrier holding the cab is released. If the path is clear, the carrier and the cab proceed in to the box transfer station to pick up the matching box (if there), or continue through the box transfer station.

10. Elevator/transfer raises in readiness for next cab from BIW -

FIGURE 3.4: TEXT DESCRIPTION OF A MANUFACTURING SITUATION.

The west loader elevator/transfer equipment transfers cabs and boxes from the BIW conveyor to the uniprime conveyor. The elevator/transfer equipment consists of four stations: 2 main stations and 2 standby stations. The first station in each pair handles the cabs, and the following station handles the box. Standby stations are located on the west side of the conveyors, and the main stations are located on the east side. The cab and the box arrive from BIW conveyor on separate carriers via the inverted Power and Free conveyor. Usually there is a box following every cab, but not always. When there are two cabs in a row, an empty box carrier is between them. The BIW delivery conveyor occupies the uppermost level parallel to and directly above the uniprime conveyor. The cab and box are transferred to a common overhead Power and Free carrier on the uniprime conveyor via the elevator/ transfer equipment.

FIGURE 3.5: SUMMARIZATION OF KAMSS UNDERSTANDING OF A SYSTEM.

3.2.3 Scenario #3 Using Sketch Input to Augment Text

If there is one true statement about collecting system descriptions from a user it is that "a picture is worth a thousand words". This is particularly true if you anticipate asking the user to type, rather than speak his input. The sketch facility of the KAMSS must allow the user to interactively create a drawing to illustrate his text. Since it cannot be assumed that the user would learn, or correctly use, a modeling formalism, the approach taken is to query the user on input of each symbol. The system will provide a standard library of symbols similar to a Apple Corporation "*MacDraw*^{TM}" system. Figures 3.6 through 3.8 illustrate the style of use of the sketch input and the interaction between the sketch and the text modes. In Figure 3.6 the user has input a series of symbols to describe the interaction of several "objects" in his environment. The purpose of the sketch input is not to enforce any particular semantics on the use of symbols but rather to "discover" the semantics of the symbols which the customer is using, and assist the user in a consistent choice of symbols. Figure 3.7 illustrates the KAMSS query of the user to force the user to characterize the "type" of object represented by a particular symbol. Figure 3.8 illustrates the integration of this sketch input with dialog input. The user has the capability to point to objects in his sketch while constructing sentences in the dialog portion of the input device. Figure 3.8 also illustrates the type of consistency checking which the KAMSS would perform on the use of symbols after they had been defined. When an inconsistency is detected the KAMSS could suggest the use of alternative symbols (different borders, shading, or even shape) for the new concepts.

Such sketch utilities would support the acquisition of facilities and product descriptions as well as function descriptions and product or data flow process descriptions. The support of such interaction requires both scope / zoom capabilities as well as abstraction and specialization capabilities.

As described in Section 8 the prototype SDCE developed provides a block diagram sketch utility which provides three symbol types, blocks, links, and groups with two line types resulting in six different usable symbols. The advantage of this approach is that generalized routing and placement support can be provided

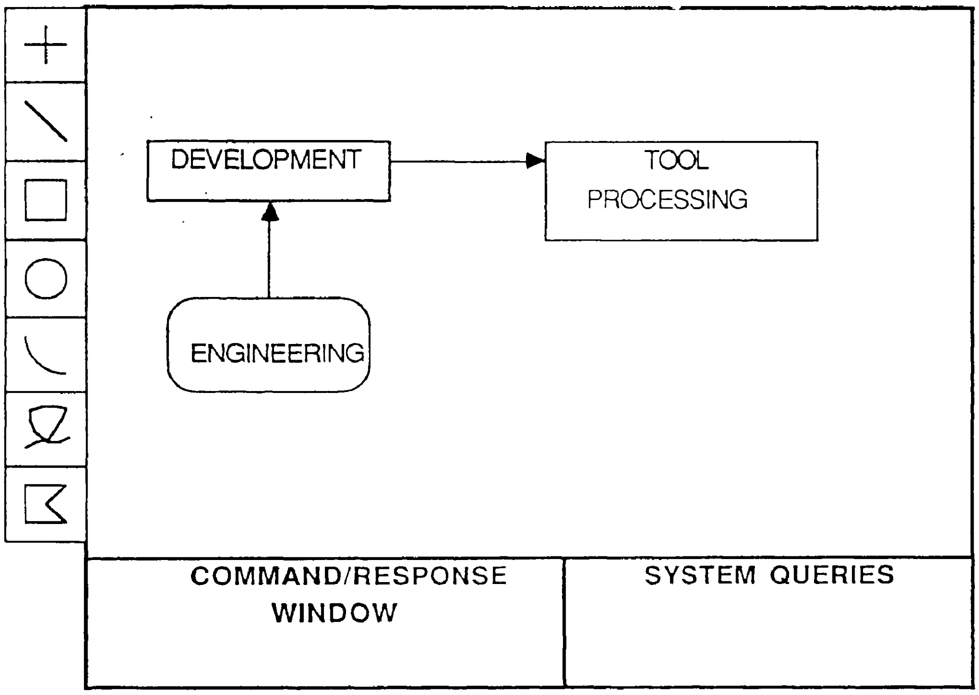
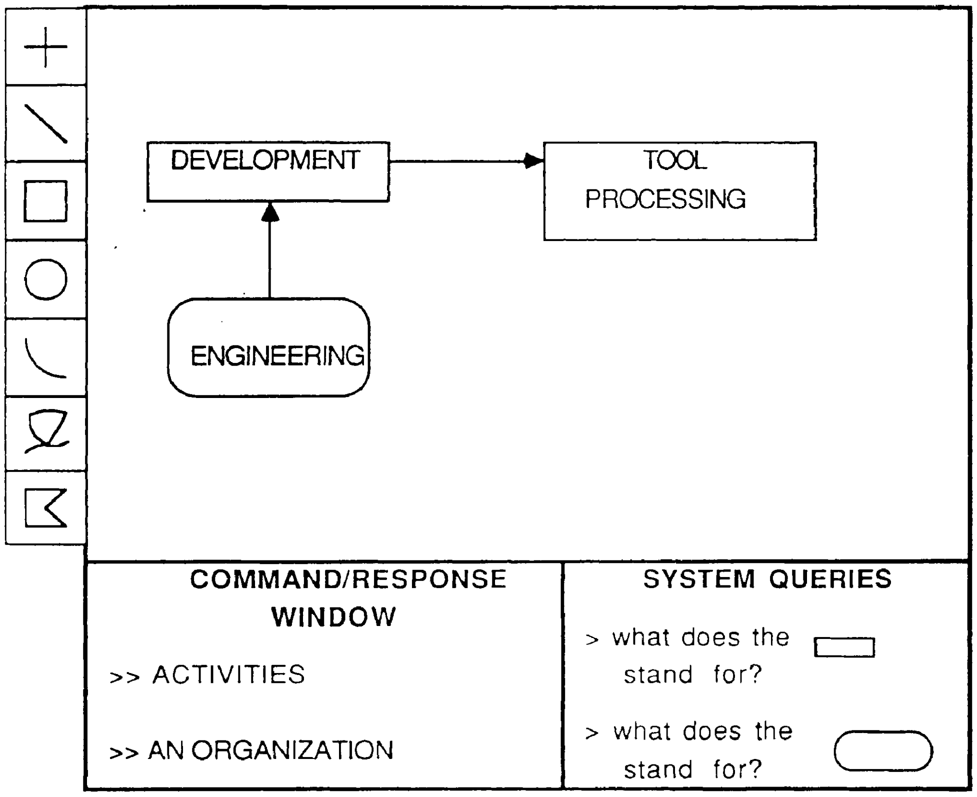


FIGURE 3.6: SKETCH CREATION IN KAMSS.



>> ACTIVITIES

>> AN ORGANIZATION

> what does the ▭ stand for?

> what does the ⬭ stand for?

FIGURE 3.7: DEFINING THE SYMBOL SEMANTICS OF A SKETCH.

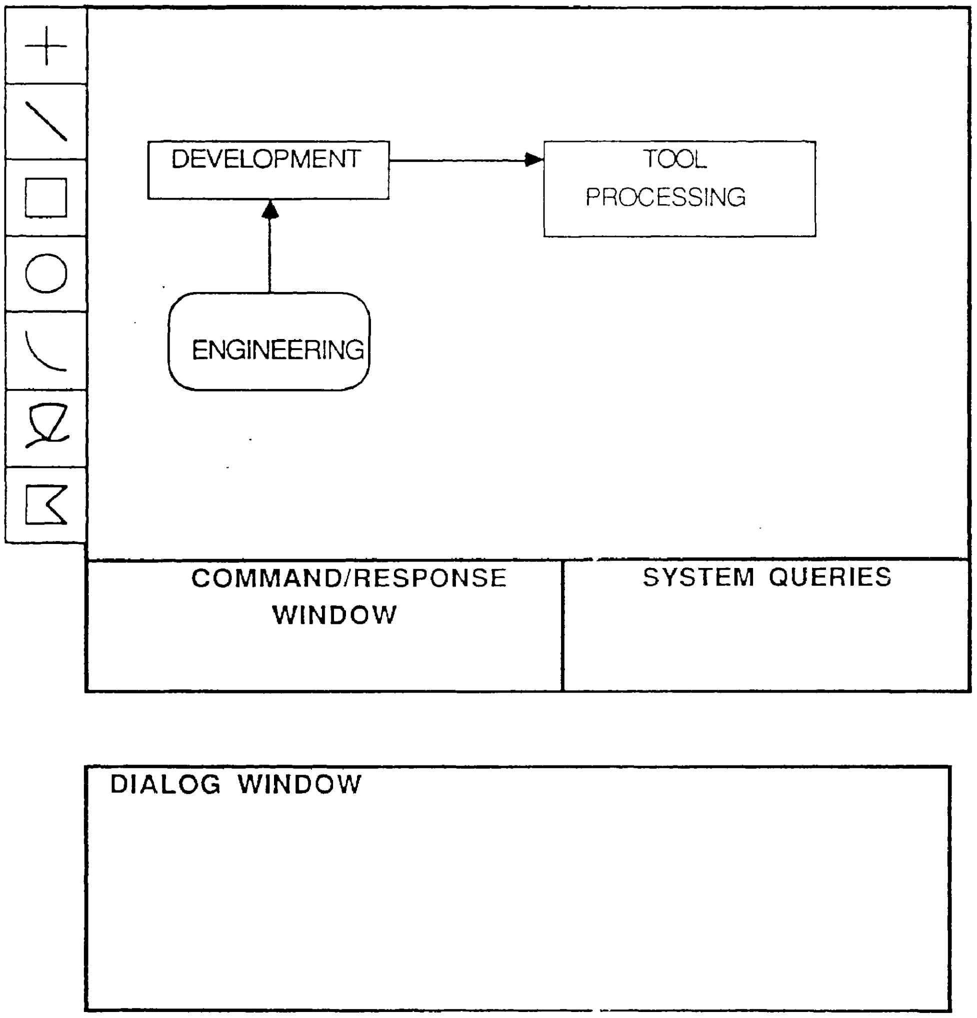


FIGURE 3.8: POINTING TO SKETCH OBJECTS AND AREAS.

allowing the user to focus on the description and not on the geometry of the illustration. In examination of the sketches used by domain experts and analysts we noted that sketches of two varieties were commonly used. The first variety (which we call the "block diagram") is used to illustrate descriptions which are independent of geometric or locational information. This first variety is also frequently devoid of timing or temporal relations (though this is less common). The second variety (which we refer to as "sketches") actually embody some abstraction of the locational relations between the components of the illustration. Thus, if two components are physically adjacent on the illustration the "adjacent" and "near" relations are intended.

3.2.4 Scenario #4 Model Design Support

By model design support we mean:

1. The ability of the KAMSS to support the creation of models which would be used for constraint based deduction.

2. The support of an experienced modeller in the construction of a simulation model for a particular purpose from a specific system description.

3. The construction of implementation models which are essentially executable applications.

Under the first scenario, it is assumed that a system description already exists and the user is accessing this database for the answer of a particular question which can be deduced from the contents of the knowledge base. However the type of question posed, or the description assertions actually available, or the type of reasoning required generally will dictate an interpretation of the description into a particular framework. This mode would assist the user in the construction of that abstraction from the description itself. The use of these type of models is discussed in further detail in the scenario on causal reasoning and qualitative simulation.

Under the second scenario the model designer would initiate the session by opening a model design window. As illustrated by this author's OBMODELER prototype described in Section 8 , the designer is presented with a set of language-specific modeling icons as well as any problem specific modeling icons which he has developed. The modeller can then ask for an analysis of the system description database in a semantic pattern directed fashion. For example, the modeller can ask for an identification of all mention of "objects which move" or the "agents of object creations". The modeller can also display the customer created sketches and query for the meaning of the various symbols displayed. Once such data has been retrieved and displayed, the modeller can use a "point and pull" method for directly moving object names, characteristics or property values from the system description to the modeling work sheet. The modeller is also provided with an "ask about" note facility which allows him to record questions relative to a particular SD object (e.g., "how many parts can be stored in front of the drill press" or "what is the average time for a purchase request to route through the signature process"). These notes can be printed out by the modeller, or they can be mailed to the customer responsible for the system description (i.e. attached to his message queue for display on subsequent login).

Since each model entity and the rules for model formulation are stored in the KAMSS system definition, the modeller can request consistency and completeness analysis as the model progresses. The modeller can also specify "probe points" in the model structure where certain attributes of entities or system states are to be collected. Traditional support for specification of the experiment, results analysis, and results presentation frames would also be required.

Under the third scenario the user would describe via the representation system the information for a particular application and rules which describe the constraints on:

1. Instantiation / acquisition of an information instance,
2. User interaction,
3. Information application logic,
4. Processing control.

From this specification of an implementation model, the user could cause an execution of the specification. In effect, the model provides a very high level programming language.

3.2.5 Scenario #5 Model Generation Support

A primary goal of the KAMSS is to provide the capability for automatic generation of models from two sources of inputs: the system descriptions and the analysis goals. Under the "Model Generation Support" scenario, the user (generally participating in the customer role) would request the system to initiate a model generation mode. The system would first have to discern what type of model the customer was actually interested in obtaining. Initially three types of models are to be supported. The first is of the information/concept structure and relation variety based on the IDEF1/ES method described in Section 8. The second is of the game theoretic variety required to perform constraint propagation. The third is of the process flow time based simulation variety as implemented in the OBSIM language. Presuming a simulation model is the desired result, the system must first interpret the information acquisition goals of the customer. The user will be queried for these goals by being presented with a menu of options which delineate the type of time persistent statistics normally associated with performance metrics of a simulation model, but independent of any specific object in the system description. Next, the model boundaries of the system must be established. There are several strategies which can be used to accomplish this task. The first is to request that the customer use the sketch facility to identify the subset of the objects in his system description which are the subject of attention in the analysis. Alternately the user can fall back on a previous sketch and using a "lasso" identify the perceived boundaries of the system in question. As another alternative, the user could request that the KAMSS attempt to "discover" the boundaries of the model by analysis of the connectivity relations between objects in the system description representation in its knowledge base.

Once the analysis objectives, information goals, and initial boundaries of the system have been established, the KAMSS model generator initiates a planning and model design process. This process uses the constraints of the modeling method to attempt to identify the required model structure and components as well as to map the system description data into these elements. The reasoning process required to support this activity is described in Section 6. A modeling theory which can be used as the basis for the constraints in this reasoning method is described in Section 7. It is expected that a large portion of the information needed to support such a model will not actually exist *a priori* in the system description data base. The user will have to be queried for such information using the results of the attempt of the system to generate a consistent token model from the system description knowledge base and the goals for analysis. This portion of the model generation process is also initiated when the user identifies to the KAMSS that certain requested information (e.g., "the average number of parts which arrive at a work cell in an hour") cannot be obtained. The KAMSS model generator must use this unavailability of information as an additional constraint to be satisfied by the model design process.

3.2.6 Scenario #6 Causal Reasoning/Qualitative Simulation

Pursuing the logical consequences of change is one of the major needs which is lacking in current decision support systems which are mathematical model based. The KAMSS must have a mode of operation in which it attempts to perform the same reasoning processes described in Section 2.2.3. We characterize this capability as qualitative reasoning support for "What If" queries. For example, at the end of the sample dialog provided in Section 2.6 the customer describes his planned modification. Presumably the human analyst would react to such a situation with questions like:

1. Which of the machines are you going to replace?
2. Will the new machines fit in the area where the existing machines are? If not, where are you going to put them?
3. What impact will such a modification have on your operation policies?
4. Will your "senior partner" be able to plan work over the new machines?

Reproducing such analyst reasoning requires that KAMSS have some built in background knowledge of the "way the physical world is" (referred to as natural constraints between situations in Section 5) and the "way typical manufacturing systems work" (referred to as conventional constraints in Section 5). Such reasoning also implies the capability of the reasoning mechanism within KAMSS to be able to apply this knowledge in a specific setting. This level of support can be provided by use of tailored proof theoretic methods given that the appropriate ontological basis is provided for the description knowledge base [Bobrow and Winograd 1977; Hobbs 1985].

The application of the knowledge included within the system description to causal reasoning and qualitative simulation requires the construction of what we refer to as a token model. The basic approach proposed for the construction of the token models involves the instantiation of model objects which conform to the type descriptions in the system description. The model object world serves as the medium for the enforcement of the nomic, physical, temporal and conventional constraints. By using type descriptors as object generators, and the axioms of change proposed in Section 5, successive states of the token world can be generated. The qualitative simulation proceeds as a constraint propagation over each object state set. The causal reasoning proceeds in parallel using pattern matching over distinguished situations.

From the user point of view the result of all three types of reasoning noted above is textual output requesting additional data, providing mechanism based explanations or descriptions of expected behavior.

3.2.7 Scenario #7 Quantitative Simulation Execution

Given that a quantitative simulation is required, once the analysis plan, simulation model, and experiment design have been completed, the next step involves the execution of the experiment using the simulation engine of KAMSS. With the natural language discourse interface, this process will be considerably different than the process of submitting simulation runs today. First of all, the KAMSS will take the direction to execute a simulation task as an activity for which the user probably does not want to sit and wait. Therefore if the KAMSS determines that the simulation run will exceed some limit, say two minutes, of real time, then it will automatically initiate the task as a batch process. The user can then continue to perform other tasks within the KAMSS architecture. The user will be able to pose questions and commands to the KAMSS as the simulation is executing such as:

1. At what stage of the simulated time is the current execution?

2. Display the number of items produced to this point in the simulation.

3. Change the priority dispatch method to the following rule.

Similar kinds of interactive simulation execution interaction (but restricted to formal command languages) have been described in [Nelson 1977; Deshler 1981].

3.2.8 Scenario #8 Interpretation of Simulation Results

As discussed in Section 2, the interpretation of the results of a simulation extend beyond the statistical analysis of the output of the simulation model. The KAMSS must support the customer or the simulation analyst in determining the implications of the model results. This includes:

1. Evaluation of the reasonableness of the model data,

2. Plan failure analysis (i.e. if the model incorporated a plan for modification of an existing operation and the results of the analysis did not show any significant behavioral change then – why not),

3. Observation interpretation, including an explanation of the behavior of the model itself.

In general, the KAMSS user would expect the system to be able to explain the "behavior" of the model itself in terms relative to the system description from which the model was generated. This behavioral explanation is critically dependent upon the availability of the token models mentioned in the previous Scenario #6.

3.2.9 Scenario #9 Decision Scenario Packaging

The customer often recognizes that a decision process which has just completed is one which will repeat itself in the future. Alternatively, one might recognize that many of his co-workers face similar decision processes. Finally, it is often the case in organizations that a path for advancement of an individual is through the systemization of a difficult problem solving activity so that it can be "delegated" to less experienced personnel under his direction. All of these conditions indicate a requirement for the KAMSS to support the "packaging" of:

1. A model implementation or series of implementations parameterized for re-use,

2. The necessary user interface to allow the specification of a new problem instance,
3. The consulting support to assist a novice in the determination of whether the scenario is applicable to the current situation,
4. The consulting support necessary to assist the novice in the formulation of the current problem,
5. The interfaces to other company databases to allow automatic acquisition of dynamic company data,
6. The consulting support to assist the novice in the interpretation of the analysis results.

Previous attempts to provide such support without knowledge based tools were limited to items #1 and #2 [Pritsker and Associates Inc. 1984]. In KAMSS, not only would the utilities for the development of such a packaged environment exist, but also the history. Using the session history management utilities, an analyst would be able to edit the interactions which the customer had with KAMSS during the process of coming up with the decision scenario. In a manner similar to a film editor, the analyst will be able to strip out the key portions of that usage scenario and use it to build a specific user interface.

3.3 User Types

The anticipated users of the KAMSS system fall into four major types. Each type of user has different support requirements, different user interface requirements and different training requirements. The four types of users are:

1. KAMSS Developers,
2. Maintenance Users,
3. Direct users,
4. Indirect users.

These types have been derived from the analysis of actual system simulation, system planning, and manufacturing engineering groups in both manufacturing system development organizations and in factory management organizations. These user types can be considered as "roles" in that a particular user can participate in many roles. A "user" need not be a human it could be another system or a subsystem within KAMSS. Each type has further subdivisions.

The first type covers the users who are defining and constructing the basic KAMSS architecture. The KAMSS Developer type breaks out into the following subtypes organized along functional orientation:

1. User Interface Developers,

2. System Architects,

3. Ontology Developers,

4. Reasoning Utility Developers,

5. Modeling Generation Developers,

6. Utility / Tool Developers,

7. Model Analysis Engine Developers,

The second type of user role covers users who are extending the built in representation structures, utilities, or analysis applications of the KAMSS. The KAMSS Maintenance User breaks out into the following subtypes:

1. Type editor users,

2. Interface Developers (to new factory or engineering information systems),

3. Packaging Developers (personnel packaging a specific decision support or description acquisition application for use by direct users),

4. Knowledge Base Administrators (maintainers of the evolving system description of a facility, and of the underlying type / class based ontology),

5. Data Base Administrators (maintaining the interfaces with the factory or engineering information systems).

The third type of user role covers those who are building the actual system descriptions and those using the system descriptions for decision making in a factory environment. The KAMSS Direct users can be segmented into two major subclasses:

1. Contributors,

2. Consumers.

Examples of contributors include facilities designers, plant engineering personnel, plant industrial engineers, manufacturing systems analysts and plant managers. Examples of consumers would include simulation modelers, production supervisors and line managers.

The fourth type of user role covers those persons who would not directly access the KAMSS but would benefit from the capabilities provided to their assistants. The primary support required for indirect users is education in the capabilities and use of the KAMSS resources. They must be made aware of the capabilities and limitations of the system.

3.4 KAMSS Architecture and Major Subsystems

The previous sections provided a general overview of the KAMSS, some typical usage scenarios, and the general philosophy of the system operations. This section will describe an implementation view of the KAMSS architecture and the major components within that architecture. The architecture presented in Figure 3.9 defines an environment which has conceptual roots in the information integrated environments of the three schema variety [International Standards Organization 1981]. Similar architectures have been implemented for the integration of manufacturing information systems [IISS 1983], engineering information systems [IDS 1987], and management decision support systems [Pritsker and Associates Inc. 1984]. There are two major differences with the concept presented in Figure 3.9. The first is that the conceptual schema component has been augmented

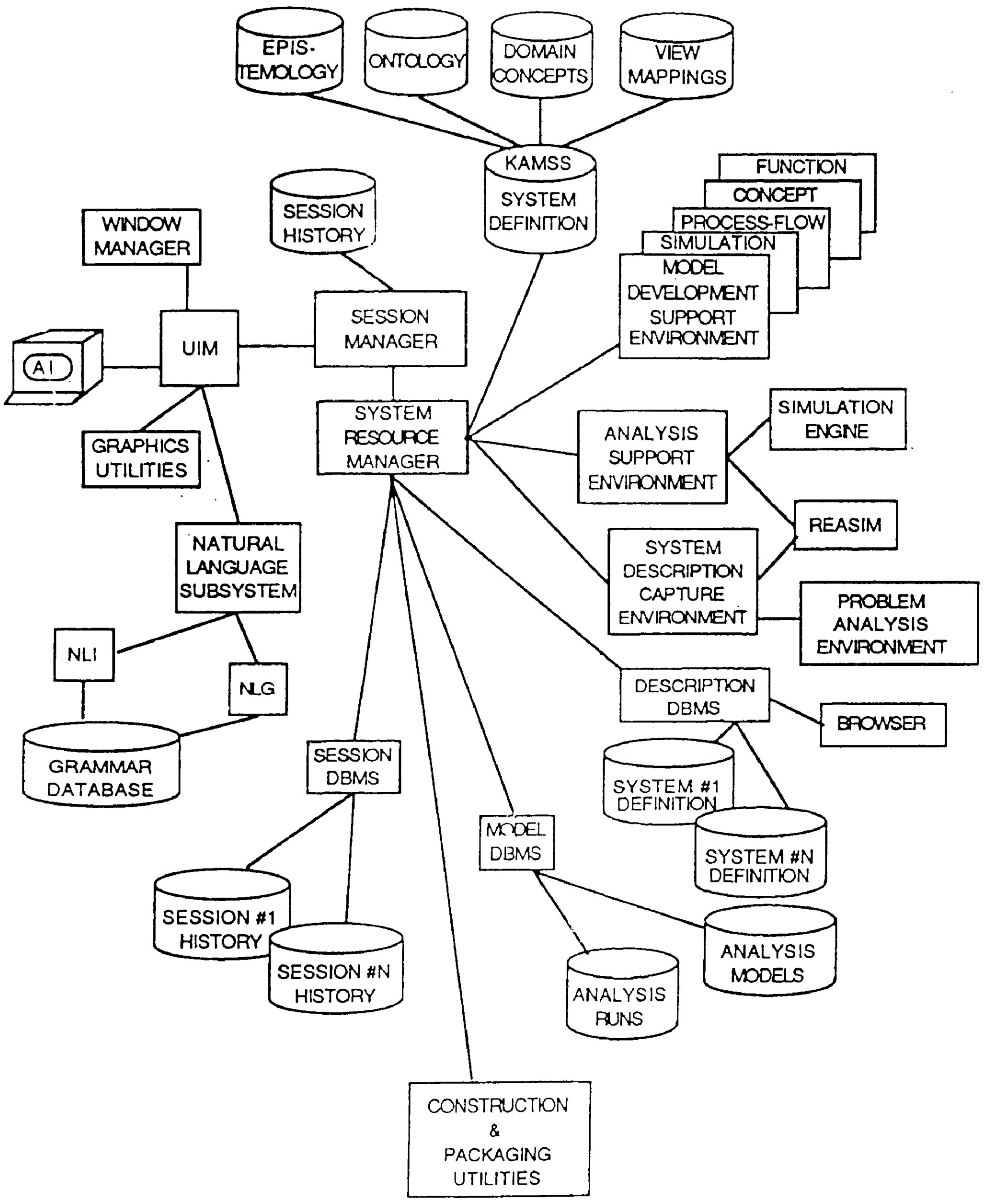


FIGURE 3.9: ARCHITECTURE VIEW OF THE MAJOR COMPONENTS OF KAMSS.

with a system definition "knowledge base". As indicated in the figure, this knowledge base contains both the definition of what information / knowledge resources exists but also the underlying specialized domain terminology, an ontology for interpretation of references and descriptions phrased in that terminology, and a set theoretic basis for that ontology. The second is that many of the elements of the environment which are integrated through this structure represent knowledge sources (i.e. individual AI applications of natural language processing, knowledge acquisition, reasoning, model generation, simulation, etc.).

The following sections will describe the functionality and structure of the seven major subsystems of KAMSS:

1. Information/Knowledge Base Management,
2. User Interaction,
3. System Resource Manager,
4. System Description Capture Environment,
5. Model Development Support Environment,
6. Analysis Support Environment,
7. Packaging and Construction Utilities.

It should be noted that these subsystems are conceived to be tightly integrated. Therefore, the individual descriptions by design will overlap to a certain extent.

3.4.1 Information/Knowledge Base Management

The information/knowledge base subsystem view of KAMSS is depicted in Figure 3.10. In addition to the distinction between knowledge bases, and information bases, the KAMSS architecture makes the distinction between stable long term, dynamic long term, private, shared, short term, and scratch memory requirements. These distinctions are made based on differences in the organization, representation, access, and management requirements of the various categories.

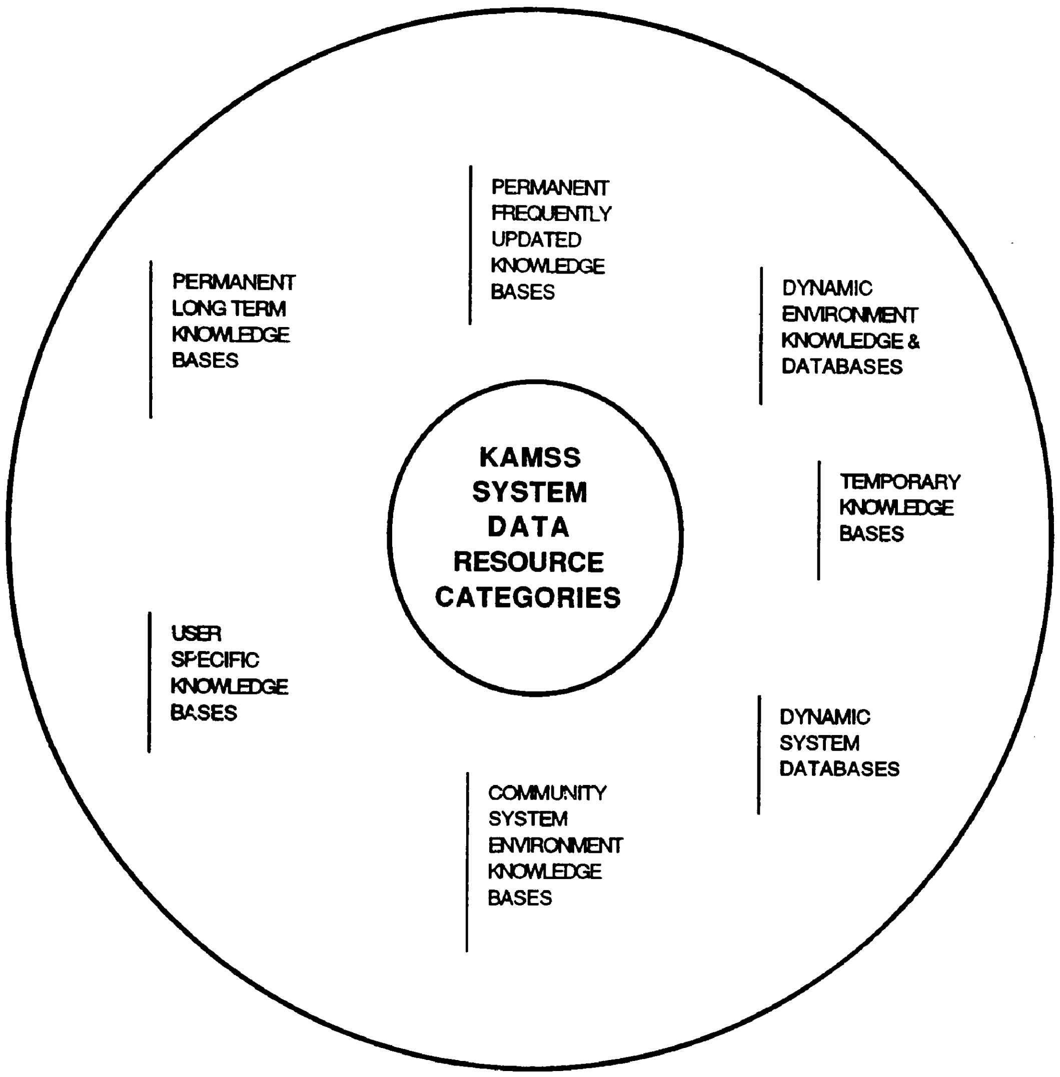

PERMANENT FREQUENTLY UPDATED KNOWLEDGE BASES
PERMANENT LONG TERM KNOWLEDGE BASES
DYNAMIC ENVIRONMENT KNOWLEDGE & DATABASES
KAMSS SYSTEM DATA RESOURCE CATEGORIES
TEMPORARY KNOWLEDGE BASES
USER SPECIFIC KNOWLEDGE BASES
DYNAMIC SYSTEM DATABASES
COMMUNITY SYSTEM ENVIRONMENT KNOWLEDGE BASES

FIGURE 3.10: INFORMATION/KNOWLEDGE BASE VIEW OF KAMSS.

Starting at the top central part of Figure 3.10 the "Permanent Frequently Updated Knowledge Bases" are those knowledge bases which exist as a permanent part of the KAMSS system but which are updated frequently. The primary responsibility for the updating of these knowledge bases would be the KAMSS system site administrator or side-effects of the "System Resource Manager" (e.g., on the introduction of a new user). This class includes most importantly the KAMSS "System Definition Knowledge Base". This knowledge base includes the definition of:

1. What system utilities exist,

2. What users are known to the system,

3. What set up is required to initiate one of the environments,

4. What data transforms are necessary to transfer data between environments, tools, or utilities.

Moving clockwise around Figure 3.10 the next class of memory is the "Dynamic Environment Information and Knowledge Bases". These structures are those created by the use of the "Modeling, System Description, or Analysis Environments" represent short-term memory in the KAMSS. Typically, they include the recording of specific usage sessions (e.g., fragments of system descriptions, generated models, etc.). These structures are maintained between sessions as private data attached to a specific user as their owner. They only influence the future execution of an environment session if they are "consulted".

The next class of memory structures consists of those which are classified as scratch memory. This class of memory structures is provided by the system resource manager to an application or environment to support the current execution. The most familiar form of this memory is the "cons" data structures which are actually managed by the underlying Lisp environment (for the purposes of presenting this design scenario, it is assumed that the KAMSS will be implemented on a Lisp machine). Beyond these structures, the most extensive user of this scratch memory is the "User Interface" subsystem.

The "Dynamic System Databases" represent a class of storage structures which contain specific time volatile data about the manufacturing system under study such as:

1. Current product mix,
2. Order status,
3. Part fabrication routings if not fixed,
4. Current production rate,
5. Operations shift schedule,
6. Current equipment status,
7. Current management goals,
8. Planning factors such as:
    8.1. Current earnings,
    8.2. Unused capacity,
    8.3. Earnings goals,
    8.4. Investment Capital and Plans.
9. External Environment Factors such as:
    9.1. Market demand,
    9.2. Current cost of money,
    9.3. Federal regulations,
    9.4. International states of affairs.

Of course, depending on the implementation environment, much of this data could be available via integration or interfacing with the corporate data bases.

The "Community System Environment Knowledge Base" is the system description knowledge base as prepared over time by many different customers or analysts. The KAMSS must provide the capability to combine private system description knowledge bases into a public combined knowledge structure. The idea behind the community knowledge bases is that these would be shared by multiple people, possibly across multiple projects, each adding incremental updates. This will allow the distribution of the cost of developing the system descriptions over many different types of projects (e.g., simulation studies, factory modernization studies, etc.). The shared knowledge bases must allow for both tentative and permanent updates. Tentative updates allow a user or project to merge private knowledge bases with the core knowledge base in an isolated environment. Other users of the knowledge base can gain access to this new extended knowledge base for purposes of validation and testing or just to browse the modifications. Once the decision to make a tentative knowledge base permanent is made, the information in the tentative knowledge base becomes the default information for all users. Shared knowledge bases must also support the accessing of previous base layers in order to allow an ease of evolution of all of the decision scenarios using the knowledge base.

"User Specific Knówledge Bases" contain information directly related to a human (or possibly other system) user of the KAMSS. This information includes the session histories, user profiles, and privileges associated with a particular user. These knowledge bases are used by the "Session Manager" and "System Resource Manager" to tailor the presentation and interaction with a particular user.

The "Permanent Long Term Knowledge Bases" store and provide access to various knowledge sources in the KAMSS such as:

1. General knowledge about manufacturing systems,
2. Knowledge about commonly used concepts,
3. The grammar databases,
4. Knowledge about how to design models, experiments,
5. Knowledge about how to plan analysis tasks,

6. Knowledge about the specific target detailed simulation programming languages.

This information is as much distinguished by the wide variety of form as by its relative permanence, in that it may take the form of procedures, rules, data structures etc., depending on usage by the KAMSS environments, or tools.

3.4.1.1 Types of Information/Knowledge

The design of the internal structures of the KAMSS knowledge bases is built on a multilevel representation scheme as illustrated in Figure 3.11. The base level contains the primitives of the epistemology (properties, bindings, and stakes). This e-level is described in detail in Section 5. This level (along with ensemble set theory [Bunt 1985]) serves as the basis for establishing a formal semantics for the rest of the KAMSS representation mechanism. The o-level contains the concepts that seem to be basic to the description of systems. They provide a basic ontology for carving up and describing that world. The method by which these concepts were identified is described in Section 4. In Section 5 these concepts are defined in terms of the e-level structures.

The d-level can be thought of as higher level macros for description construction that are specific to the manufacturing *domain*. This level contains more domain specific packages of o-level structures. It exists for the purpose of narrowing the "semantic" gap between the user input and the computer storage of that input in a form which can be reasoned with by the computer. Because of the complexity of the manufacturing domain we found it necessary to provide additional structure to the d-level as illustrated in Figure 3.12. This additional structure takes the form of sublevels and views. The use of this additional structuring will be described in the section on the System Description Capture Environment (SDCE) found later in this section and further elaborated in Section 8. As there must be a language designed for talking about each of these levels the s-level contains the symbols and structures for accomplishing that task.

Finally the m-level contains the concepts of the method for semantic model development. This includes properties, bindings, stakes and the mappings of these classes of objects to the primitives in the e-level. This level is used as a reference for the processing of new referents which have not been (or cannot be) indexed

M-LEVEL

E-LEVEL	0-LEVEL	S-LEVEL	D-LEVEL
Properties	Actions	Prototype	Machine
Bindings	Objects	Schema	Material Handler
States	- discrete	Slot	Part
	- continuous	Relation	Product
	- composite	Constraint	Process
	Time	Measures	Data
	Space		Plan
	Quantities		Procedure
	Situations		
	Events		
	Processes		
	Conditions		
	Causality		
	Sequence		
	Iteration		
	System		
	Subsystem		
	Component		
	Operation		

FIGURE 3.11: LEVELS OF INFORMATION/KNOWLEDGE REPRESENTATION IN KAMSS.

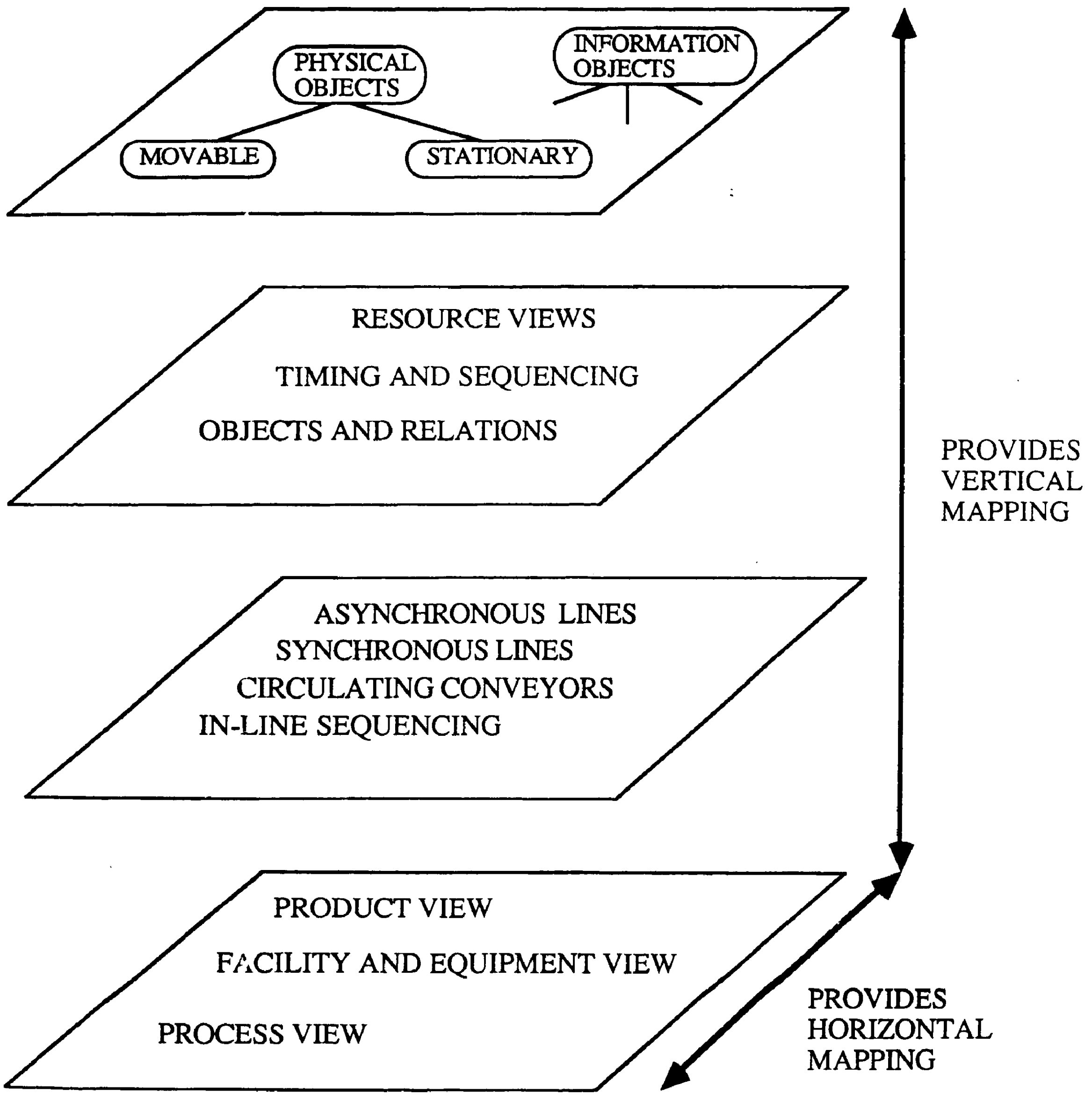


FIGURE 3.12: LEVELS AND VIEWS IN THE DOMAIN LEVEL.

into one of the other levels. It represents the minimal information that can be known about the referent of the use of a symbol in the system.

3.4.2 User Interaction

The "User Interaction" subsystem includes the "Session Manager", "User Interface Manager (UIM)", the "NL Utilities" and the "Hyper-Notes" utilities as shown in Figure 3.9. This subsystem also contains the utilities for classification support and online tutorial / help support. One of the issues which needs to be addressed is the question of why the need for such a technologically complex user interface. Why not just use a set of input forms and a command language input? The justification that we offer is based on experience in using and developing such systems. Take for example the ISDOS [Teichroew and Hershey 1977] or SREM [Alford et al. 1979] system requirements definition systems or the IDSS 2.0 [Pritsker and Associates, Inc. 1983]. Just remembering the forms or language constructs which are available and which should be used to specify a certain type of definition or relationship is a challenging task itself. For a system like KAMSS, the number of different forms would be significant (estimated at over 500). To plan to only provide these as a form of interface would be tantamount to retiring the system before it is even designed. It is our belief that even a limited natural language capability (as demonstrated in the SDCE and MODGEN prototypes) can effectively be substituted for the form input without extreme resource penalty and with substantial increase in system usability.

One criticism of this approach is that it may require excessive typing on the part of the user. This criticism is one to take seriously. However, the use of spelling correction, interactive processing with FIX and DWIM features [Teitelman and Masinter 1981] can substantially reduce the negative impact. Also, there are two other avenues which can be pursued to reduce the amount of specific input which is required. One avenue is to build more knowledge into the KAMSS about what one normally finds in manufacturing system. The second is to provide the user with the capability to enter sketches and then refer (by pointing) to these sketches. Both of these approaches have been incorporated into the KAMSS design. The first is considered the domain of the specific "Environment" subsystems and will be discussed in those respective sections. For example, in the SDCE domain, it is expected that the major mode of text input will be in the editing of existing text. Thus, for example, a user can describe a mechanism by choosing a similar mechanism and editing the textual / graphics description of that mechanism. The existing text has not only been parsed but a semantic representation has already been built. The text itself is composed of specialized objects which "present" themselves as what appears to be text. These objects "know" how they can be modified and how to parse and interpret a modification. The user can also deal with these descriptions strictly at the "type" level. That is, a user can declare the new mechanism as a subtype of an existing mechanism. He can also specify the nature of the subtype relation (e.g., inheritance of physical characteristics, or behavioral characteristics). The latter is considered a part of the user interaction subsystem.

The combination of these capabilities with the ability of the user to use text and free hand sketches as input mechanisms means that, besides the processing of the respective types of inputs, there is the added problem of attaching semantics to the symbols used and the correlation of the text to the graphics. Figure 3.13 displays the major components of the "User Interaction" subsystem. The "discourse manager" is the module responsible for monitoring the actual dialog and performing the mode sensing and integration. The mode sensing can be accomplished reasonably using the utilities of the Symbolics window and process systems. The task which adds complexity to the discourse manager is that of "strategic" response planning.

Generation of natural language text can be viewed as a two stage process; the strategic planning phase determine "what" should be said, and the tactical planning stage determines "how" to say it. In the architecture presented in Figure 3.13, the application and the discourse manager determines "what" will be said, and the discourse generator determines "how" it will be said. This decomposition of functionality represents adherence to the principle that "there is no such thing as generation in the abstract: one must study the generation of specific, well-developed artificial speakers performing in specific discourse contexts" [McDonald 1982]. The discourse manager and the discourse generator encompass the situation and linguistic knowledge respectively, leaving the domain knowledge to be a part of the particular application environment. Section 4 provides a discussion of the proposed text generation approach.

In the prototype SDCE, the use of type sensitive case structured input was substituted for a full unrestricted natural language input. This decision was made for two reasons. First of all, the requirements efforts indicated that the direct users preferred such an interface since it minimizes the typed input required on their part. By using the facilities provided by the presentation types, command completion, and special handlers the system can take advantage of its knowledge of the input situation to anticipate or allow user selection of input sentence components from a menu. This significantly reduces the effort and chance for error. It also supports a form of knowledge acquisition which has proven effective in constructing large scale knowledge bases, that being the "Copy and Edit" approach [Bobrow and Stefik 1983; Lenat et al. 1986]. The second reason was to reduce the complexity and achieve reasonable performance in the prototype.

The "concept manager" indicated in Figure 3.13 provides a level of flexibility in the user interface to allow for the use of a minimal lexicon processing strategy as described in Section 4. As each sentence is processed by the parsing component the syntactic indicators of types, entities, or constraints (as per the IDEF1/ES analysis see Sections 4 and 8) are used to perform an extraction of these elements. These extracted structures are passed to the concept manager which checks for their existence in the current concept knowledge base. The concept manager attempts to classify the new concept based on the current discourse, the syntactic structures used and the existing concepts in the concept data base. If it can successfully classify the concept, then it updates the concept base. If it cannot, then if the invocation was the result of a successful parse the concept manager delays the resolution of the problem until at least three additional utterance inputs have been processed. If the invocation was the result of a parse failure, then the concept manager will invoke the discourse generator with a planned yes/no question centered around the constraint portion of the utterance (the apparent verb phrase). On receipt of a successful answer to the question, the concept manager will recommend an update to the case frame structures in the grammar database. It is the responsibility of the discourse manager to then update the grammar database structures. Section 5 provides a description of a theory of semantics which can support this concept discovery approach. That section also includes a more detailed example of its application to the minimal lexicon problem.

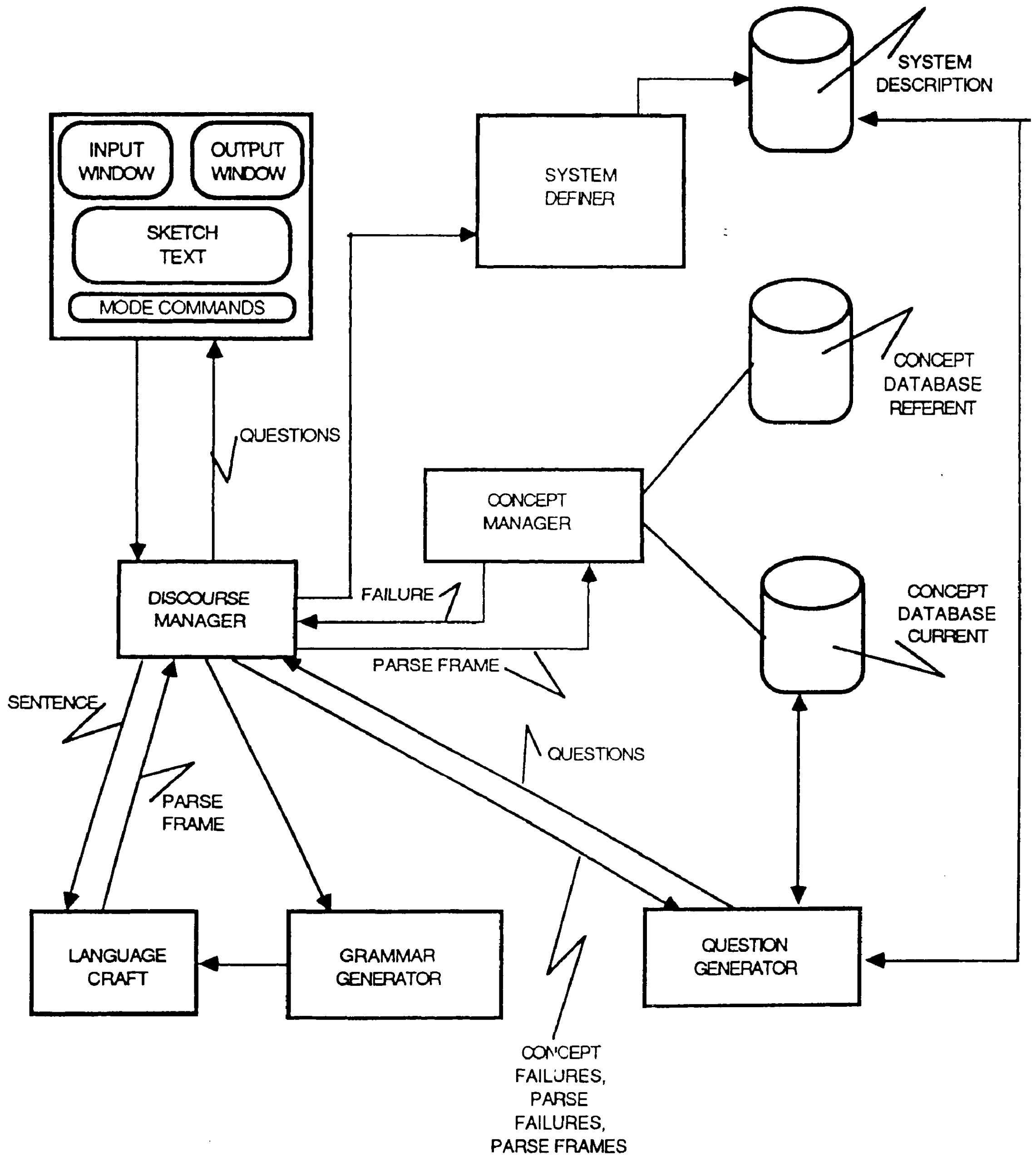


FIGURE 3.13: USER INTERACTION SUBSYSTEM.

This cooperative use of the concept manager, classifier, discourse generator, and discourse manager allows the KAMSS to operate initially with a minimal lexicon and also a minimal set of case frame grammar structures. In a very real sense, this capability amounts to an acquisition of knowledge at the "buzz-word" level (see Section 5).

3.4.3 System Resource Manager

The system resource manager controls the invocation of major modes of processing as a traditional "monitor" component would. In addition, it has an expanded role which includes the managing of access to the various information / knowledge bases. Management of access includes more than verification of user privilege. As indicated above, the KAMSS includes a large number of data sources. Some of these are information sources in a traditional database organization. Others are "knowledge bases" of rules, schema structures, and procedures. Thus, for example, the grammar structures may be encoded in Lisp structures organized in a hashed table, the session histories may be stored as formatted annotated flat text files, the model design knowledge bases may be ART^{TM} rule sets, where the system descriptions are stored on a VAX using the Knowledge Craft^{TM} schema base manager. In order to have the design flexibility to use the best physical management software for each different type of data resource, one needs a layer of management that knows "where everything is" and how to translate between the different forms of representation. This is the primary function of the system resource manager. As in the typical three-schema architecture approach to this problem [International Standards Organization 1981], the system resource manager accesses the definitions of what exists (in what form and where) from the system definition knowledge base.

3.4.4 System Description Capture Environment

Manufacturing and production system descriptions themselves are complex knowledge bases which include information about:

1. Manufacturing facility layout, location, age, current product mix,

2. Equipment and tooling, with their associated characteristics,

3. Business rules and operating policies,

4. Manpower levels and skill classes,
5. Union and government regulations,
6. Information: its structure, flow, and use,
7. Plans, goals, schedules, and commitments of the organization,
8. Situations, events, and courses of events to which describe the dynamics of the system,
9. Relations which can be established between the above types of information and the constraints which those relations conform or impose.

The capture of such descriptions and subsequent use or display must take into account a number of different "viewpoints" including:

1. The corporate strategic planning view,
2. The product strategic planning view,
3. The product definition view,
4. The manufacturing planning view,
5. Individual facility strategic planning views,
6. The organizational view,
7. The physical facility view,
8. The operating personnel view including:
   - 8.1. The general manager view,
   - 8.2. The area manager view,
   - 8.3. The foreman view,
   - 8.4. The operator view,

   8.5. The manufacturing engineering view,

   8.6. The quality control view,

   8.7. The industrial engineering view,

   8.8. The union view.

9. The process view,

10. The information view,

11. The modeling view.

The implication of these viewpoints on the representational scheme is that it must be able to represent and accommodate partial knowledge about an entity from multiple descriptors that describe the same entity from different viewpoints. The complexity of viewpoints and the desire for a concept discovery capability underscores the need to use a flexible underlying representation scheme which does not over commit to the prior representation of specific areas of knowledge.

Previous attempts to capture such complex situations have failed both because they required the users to learn and to use complex unnatural specification languages, and because they attempted to utilize traditional rigid data base techniques for the storage of the descriptions. For these reasons, we are proposing to use a combination of form, menu, natural language text, and picture sketch input combined with a knowledge based management tool for management and manipulation of the descriptions. We are also proposing a basic semantics representation and syntactic based reasoning strategy (see Sections 5 and 6) which can accommodate the free objectification which humans use in their formulation of system understandings as described in Section 2. Finally, the proposed design relies heavily on the reusability of existing descriptions as a primary means of information acquisition. Thus the *copy and modify* philosophy is supported in each mode of input.

Figure 3.14 displays the basic structure of the system description capture component of the KAMSS.

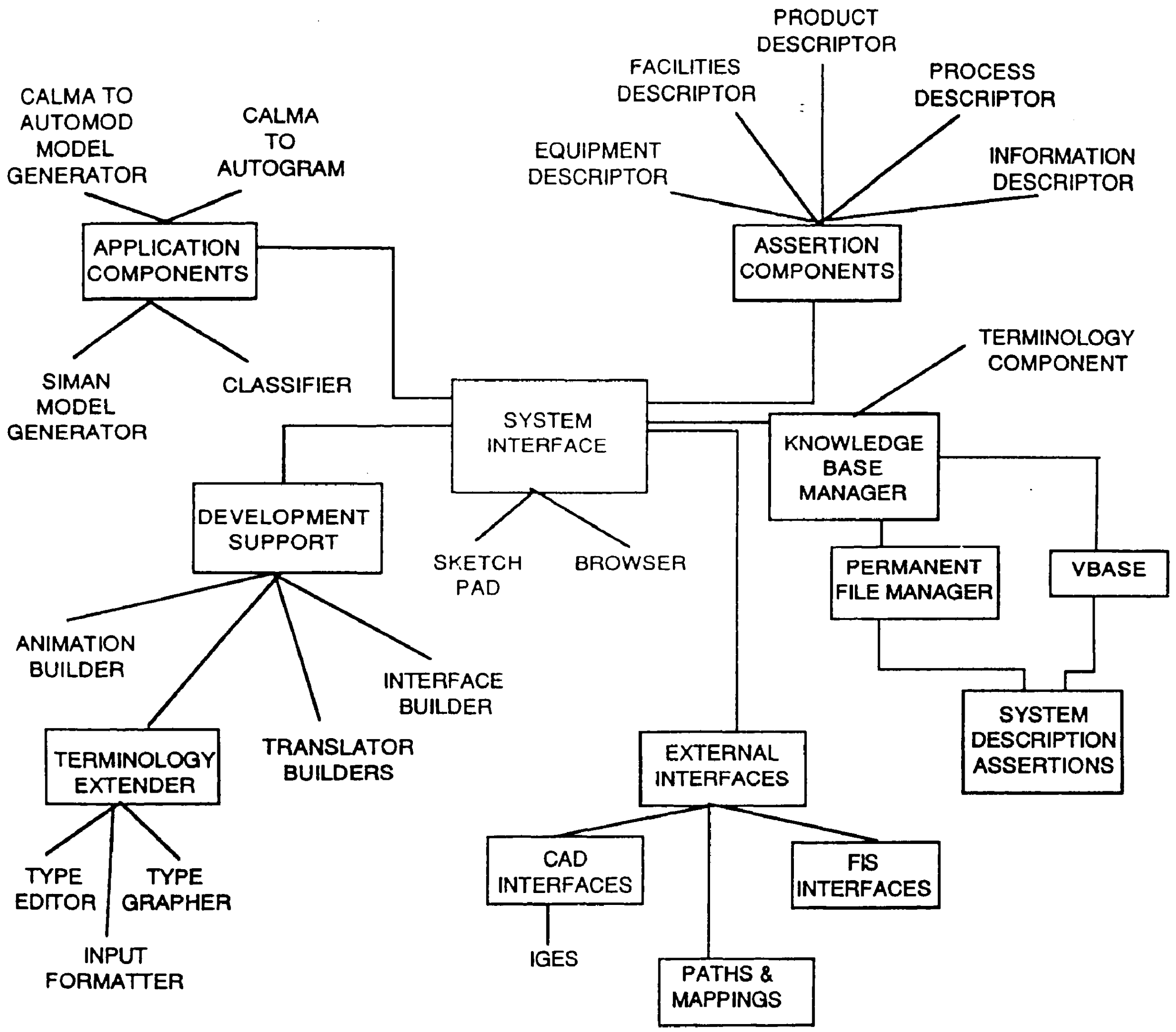


FIGURE 3.14: SDCE ARCHITECTURE.

3.4.5 Model Development Support Environment

The model development support environment depicted in Figure 3.15 provides the following functionality:

1. Analysis Planning, including:
    - 1.1. Determining the questions that must be asked to determine a particular model usage goals and context,
    - 1.2. Design of experiments,
    - 1.3. Production of a costed and timed model plan,
2. Model Design, including:
    - 2.1. Extraction of IDEF1 models from the system descriptions,
    - 2.2. Qualitative simulation model design directly from the system description,
    - 2.3. Extraction of object flow models (where an object can be a part of a specific type of information).
3. Quantitative Simulation Model Design, including:
    - 3.1. Design of simulation model mechanisms.
    - 3.2. Direct use of system description as constraints on simulation mechanism design.
    - 3.3. Generation of detailed model specification in a simulation language.
4. Implementation Model Design (using the KAMSS as an engineering programmer's workbench) which includes:
    - 4.1. Use of the KAMSS framework as a platform for building specialized applications,
    - 4.2. Object definition code generation from a type description,

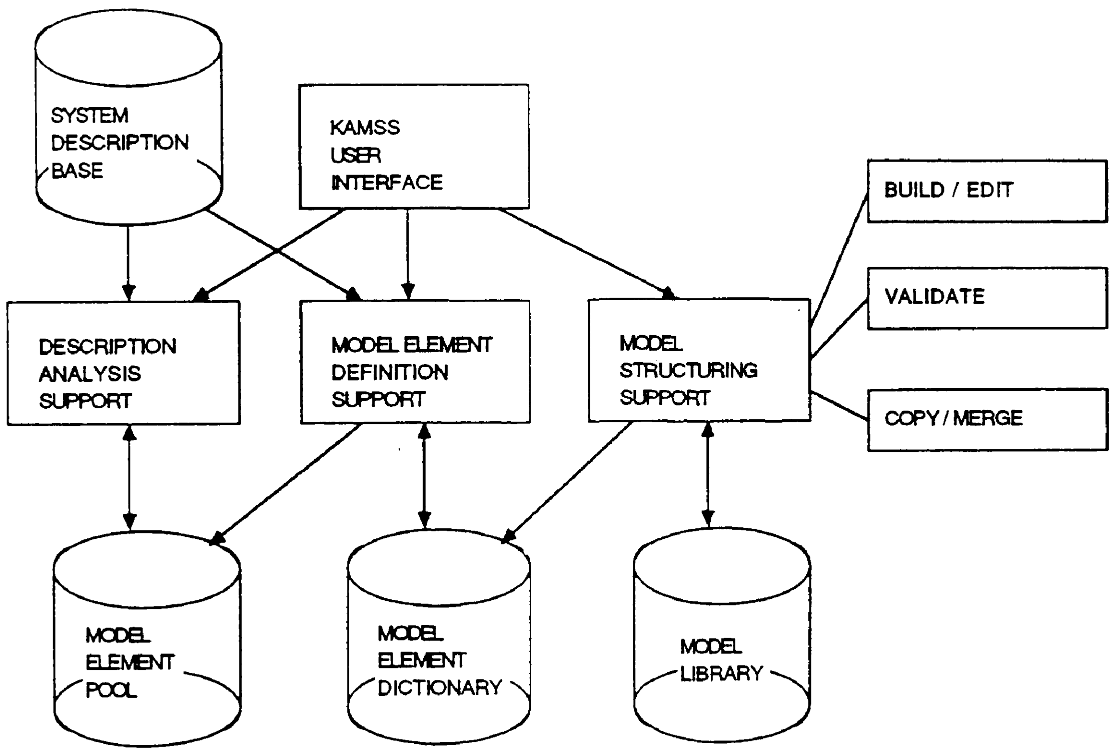


FIGURE 3.15: MDS ARCHITECTURE.

4.3. Lisp method code generation from a rule language specification of processing behavior,

4.4. Relational schema generation for data acquisition prototypes from class descriptions,

4.5. Procedural code generation from functional diagrams.

3.4.6 Analysis Support Environment (ASE)

The Analysis Support Environment (ASE) provides a framework to support many different types of analysis methods. As illustrated in Figure 3.16, this includes the capabilities to perform causal reasoning and qualitative analysis of the system descriptions as well as mathematical simulation of the system models. The ASE also provides the structures for housing additional analytic techniques or packaged decision scenarios. As a minimum, the KAMSS ASE will provide:

1. Qualitative reasoning about system dynamics from system descriptions,

2. Prediction of the implication of questions,

3. Qualitative simulation,

4. Simulation experiment processing,

5. Simulation experiment results interpretation,

6. Explaining simulation predicted system behavior.

3.4.7 Packaging and Construction Utilities

The major utility subsystems for the KAMSS are illustrated in Figure 3.17. These include the utilities for:

1. Decision scenario packaging,

2. Forms generation,

3. IGES/PDDI graphics interface construction,

4. Knowledge base maintenance,

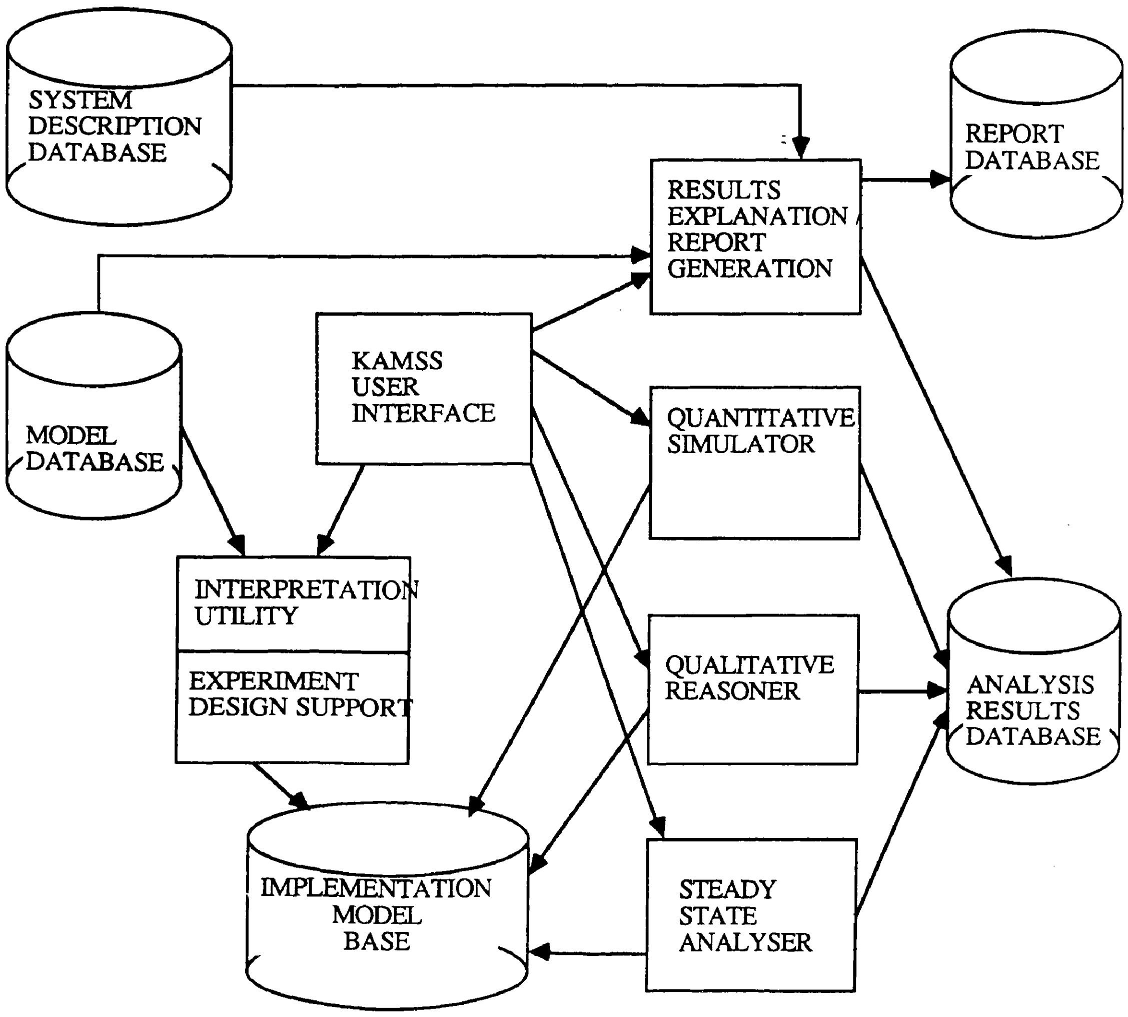

SYSTEM DESCRIPTION DATABASE
REPORT DATABASE
RESULTS EXPLANATION / REPORT GENERATION
MODEL DATABASE
KAMSS USER INTERFACE
QUANTITATIVE SIMULATOR
INTERPRETATION UTILITY
EXPERIMENT DESIGN SUPPORT
QUALITATIVE REASONER
ANALYSIS RESULTS DATABASE
IMPLEMENTATION MODEL BASE
STEADY STATE ANALYSER

FIGURE 3.16: ASE COMPONENTS.

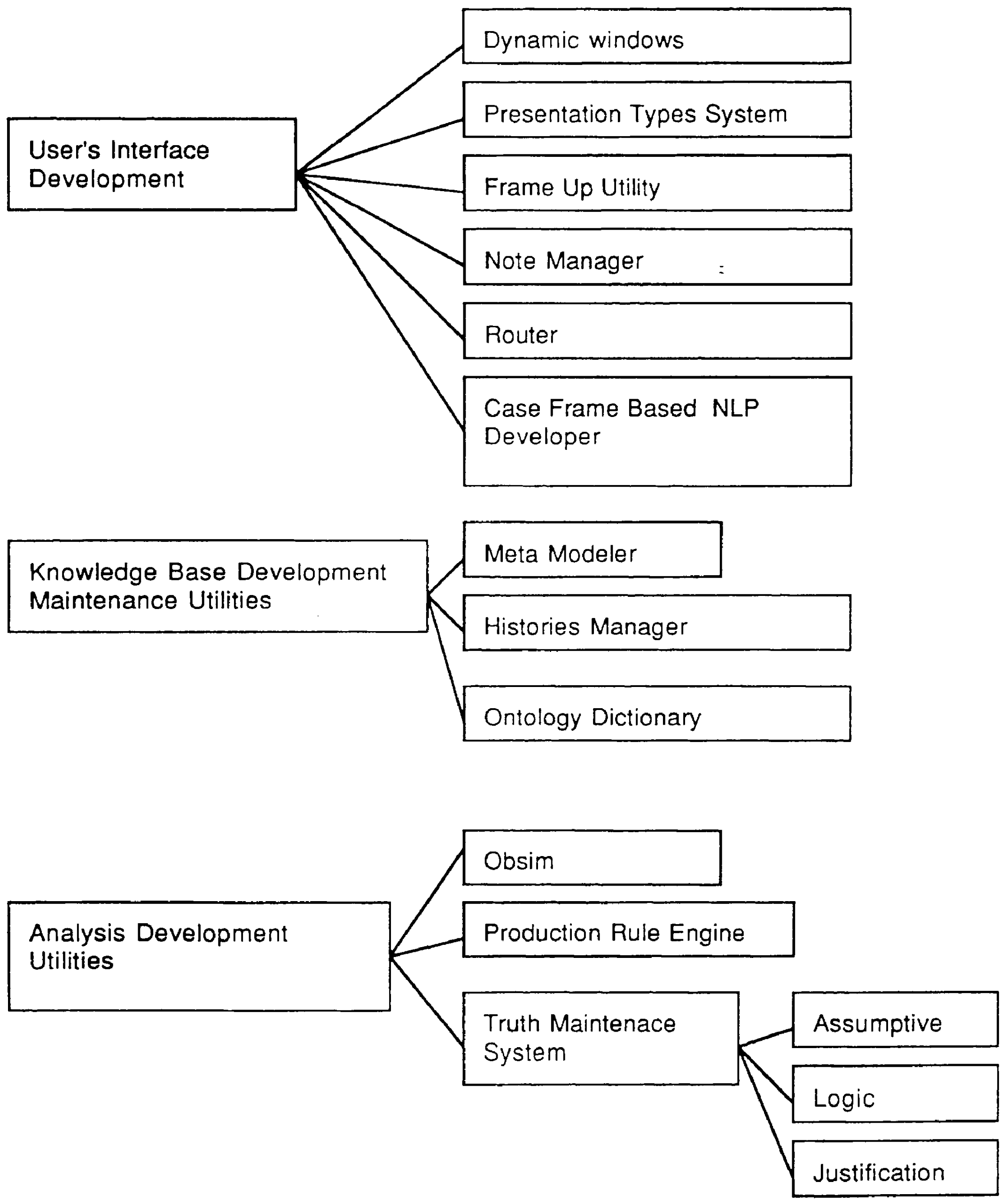


FIGURE 3.17: KAMSS PACKAGING AND CONSTRUCTION UTILITIES.

5. Graphical model layout and connection routing utilities,
6. Model Data Base generation,
7. Data base administration,
8. Factory Information System interface construction.

The utilities are supported by a development environment modeled after the Knowledge Craft^{TM} Toolbox concept. Under this concept each utility is set up to have a common set of access, help, and usage command structure. This is possible because of the assumed level of sophistication of the user of this environment.

3.5 Implementation Issues for KAMSS

The implementation of the KAMSS architecture presented in the previous sections has several major theoretical and practical issues associated with its construction, which are the focus of the remaining sections of this dissertation.

1. The lack of precise cognitive models of all the reasoning activities involved, or of their placement in a classification system of reasoning types,
2. The current lack of an adequate theoretical semantics basis,
3. The lack of an available reasoning method powerful enough to handle the variety of reasoning processes required,
4. The complexity of the natural language processing requirements,
5. The size and complexity of the resulting knowledge bases,
6. The lack of an available simulation engine to process the simulation models of the complexity and size of those which will be generated by the system.

3.5.1 Overview of Existing Tools and Their Applicability

The obvious preference for an implementation approach to KAMSS would be to build from an existing commercial knowledge engineering environment. This has advantages in terms of long term maintainability and possible commercialization of the KAMSS concept, as well as avoiding the recreation of a product which represents hundreds of man-years of development. The disadvantages include the possible need to integrate several of these tools to construct a system of the complexity outlined above.

Eight commercial knowledge engineering / object management environments have been evaluated as platforms for the construction of the KAMSS. These include:

1. Knowledge Craft^{TM} (Carnegie Group Inc.),
2. Language Craft^{TM} (Carnegie Group Inc.),
3. ART^{TM} (Inference Corp),
4. KEE^{TM} (Intellicorp),
5. LOOPS^{TM} (Xerox),
6. Vbase^{TM} (Ontologics),
7. Joshua^{TM} (Symbolics),
8. Statice^{TM} (Symbolics).

A complete evaluation of the characteristics of these tools can be found in [Mayer et al 1986; Mayer 1986]. The following paragraphs summarize the conclusions of these evaluations relative to the KAMSS architecture.

First of all, one of the implicit requirements for a knowledge engineering tool is the ability to have a data management capability for the storage of permanent objects. This requirement all but eliminates the ART and KEE systems from consideration as the dominant structure for the KAMSS architecture. Knowledge Craft, LOOPS, and Vbase all provide a database capability for the storage of permanent object definitions. Of these three, Vbase and Knowledge Craft stand out as the most promising. LOOPS is not yet a supported product and it is also limited to execution on the Xerox 1108/1186 workstations. Strobe (Sun Microsystems) provides most of the same capabilities of LOOPS and it executes on the Sun environments as well as the Xerox environments. Unfortunately, the drawback of either the Strobe or the Sybase products is the limited knowledge representation which is available in the underlying language.

The best choice of a base structure for the KAMSS architecture appears to be to use the Knowledge Craft [Carnegie Group 1987] system as the basis for the representation mechanism for the description capture environment and the model generation environments. The power of the schema representation language and the ability to describe the semantics of relations in Knowledge Craft is clearly superior to the other available systems [Mayer et al 1986; Mayer 1986]. With the availability of a schema database mechanism (available on the VAX and Sun versions), the Knowledge Craft tool appears to satisfy the basic requirements for the semantic structures and reasoning mechanisms as well as the practical knowledge base management services required to support KAMSS. However, in the prototype implementations of the components of KAMSS the non-uniform treatment of attributes of frames by the Knowledge Craft tool complicated the development of a knowledge base to support an evolving description to the point where we chose to develop our own object management system. The results of our preliminary evaluation of the Joshua and Statice environments are such that we intend on evolving to these systems as they become available.

Hardware Implications of KAMSS:

During the course of this research and the development of the prototype implementations, we have experimented with expert system development tools on Symbolics 3600 machines, Texas Instruments Explorers, Xerox 1108 machines, Digital Equipment Corporation VAX 11/7XX machines, as well as IBM PC/AT and Apple Macintosh micro computers. The purpose of this section is to examine the hardware implications of the KAMSS architecture and assess the role of Lisp machines, multi-user traditional machines, and micro-computers as well as networking requirements to support the development and use of the KAMSS.

In its final realization, the KAMSS will almost certainly exist on a heterogeneous set of hardware. There will be need, for example, for information acquisition support utilities which operate on portable devices which the analyst can take to the field with him. There will be need for multi-user access to the system description knowledge base by manager decision makers to pose questions to the knowledge base. There will be need for special purpose hardware (parallel architectures) for running the size of simulation models which will be generated with the system. And there will be need for the sophisticated AI workstations for support of the analysts, modellers and knowledge base / database maintenance personnel. Figure 3.18 illustrates a possible hardware implementation of the KAMSS.

3.5.2 Prototypes: Functionality and Rationale

The following prototypes were constructed during the course of this research as proof of engineering prototypes for the KAMSS architecture and to establish some of the critical methods and tools for a full scale implementation of the KAMSS architecture:

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 Tool (FCT),
5. Model Development Support Environment Prototype (MDSE),
6. Qualitative Reasoner from System Descriptions (QUARS),
7. Model Generation from System Descriptions (MODGEN),
8. Situation Based System Description Knowledge Database (SDKD),
9. Model Design Support from System Description Sketches (OBMODLER),
10. An extensible, object based, simulation engine (OBSIM).

These prototypes are discussed in detail in Section 8 of this dissertation. However, it is useful in this section to review the portion of the above described design concepts they were designed to illustrate.

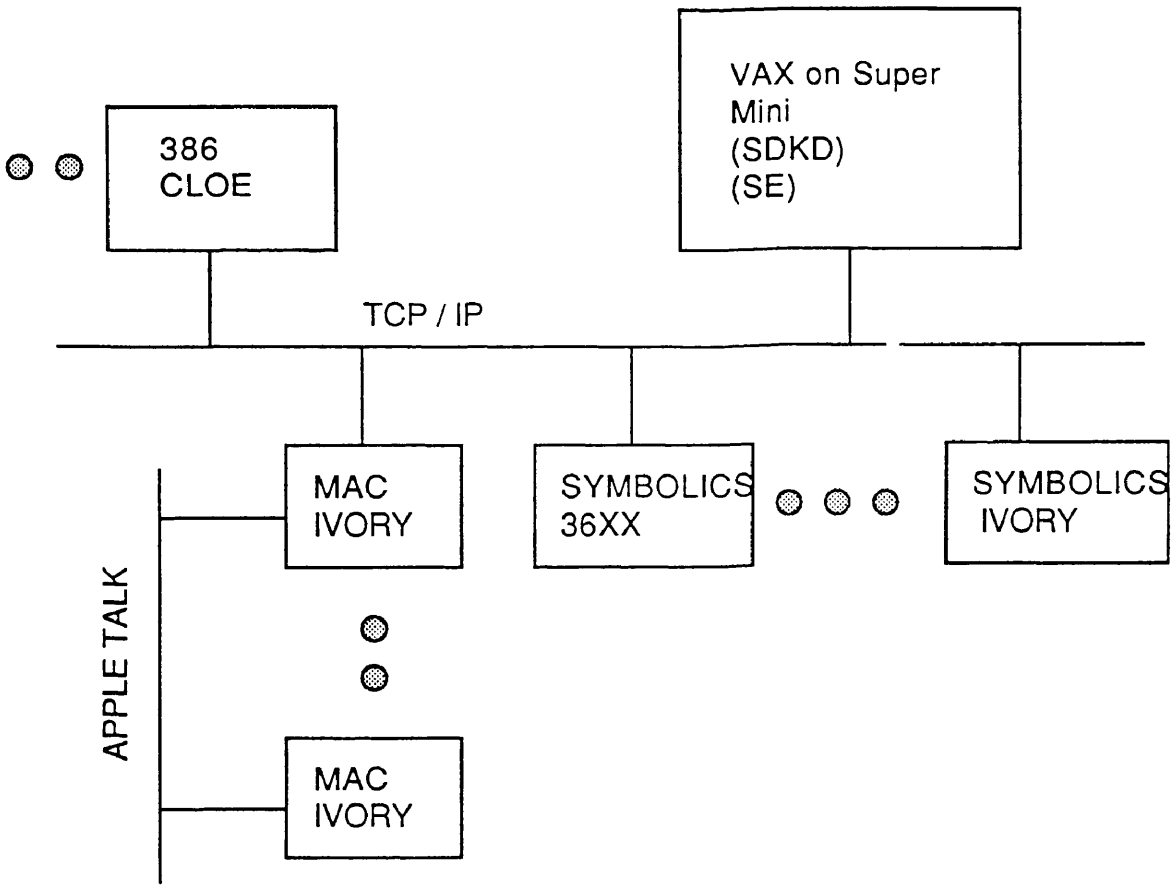


FIGURE 3.18: KAMSS IMPLEMENTATION ARCHITECTURE.

The IDEF1/ES methodology was designed, based on the method for semantic theory development described in Section 5, as a tool to be used to construct the ontologies needed for the design/evolution of the permanent long term knowledge bases described above. Since the base from which this tool was evolved is the IDEF-1 methodology [Ramey 1983], it has also been useful for the definition of the requirements for the other information and knowledge structures in KAMSS.

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 high priority need was identified 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.

The SDCE was designed to test the system description knowledge representation scheme, intelligent user interface, form and sketch input, browsing, hyper-text and CAD/FIS interfaces presented previously. This prototype features both a concept sensitive structured diagram interface and concept description capability and hence serves as a prototype of the dialog processing and management features of KAMSS.

The FCT was designed to experiment with automated support for the collection of facts which describe situations in a organization environment using a natural language input mechanism. The limitations of the technology for direct processing of natural language (particularly in the micro processor environment where we wanted to delivery this capability) led to an approach where the user guides the semantic interpretation of the statements entered.

There are actually two MDSE prototypes. The first MDSE prototype is focused on the provision of model development, integration, and validation support for the construction of IDEF0 and IDEF1 models. The other is being built on top of the SDCE prototype using the block diagram editor and object editor to support the design of simulation models based on the SIMAN simulation language constructs.

QUARS was designed to demonstrate the feasibility of applying the syntactic information chain reasoning concepts of the AR theory (see Section 6) to system descriptions whose representation was consistent with the ontology presented in Section 5. This prototype actually consists of two components. The first performs causal reasoning based on a tokenization approach. This component includes a form of temporal constraint propagation with an assumptive based truth maintenance scheme to support evaluation of the feasibility of the control logic of a manufacturing process directly from an SDCE description. The second is built off of the "Q" system and provides for qualitative simulation of continuous process models generated from the system descriptions.

MODGEN was designed to demonstrate the feasibility of directly generating simulation models from system descriptions captured using the SDCE. The prototype developed used description structures similar to those found in such simulation languages as MAP-I^{TM}. The prototype was limited in its ability to handle conditional constraint specifications and in its ability to handle a variety of question types.

The SDKD was actually prototyped as a part of the SDCE, where the primary issues were representation completeness and access efficiencies. The second prototype will be a stand alone implementation (one in Statice and the other in Vbase) concerned with the issues of efficiencies for large description databases and distribution of such data bases.

The OBMODLER was initially constructed in the INTERLISP LOOPS environment for the purpose of exploring the issues of free sketch input of system description pictures and the use of these pictures as an aid to model design. However, the concepts of this model development support environment were subsumed by the SDCE.

Finally, the OBSIM was constructed to test the direct simulation of object specification of model components, and the specification of business operation policies in a rule form. This prototype was initially constructed using LOOPS on the Xerox 1108 AI workstation. The architecture of this system is described in detail in Section 8 of this dissertation. It provides a simulation language development environment as well as a basic simulation engine. Within the OBSIM environment a modeller can mix simulation modeling paradigms or construct a unique one for a particular problem. By capitalizing on the object oriented programming paradigms the OBSIM is easily extendible / adaptable to the simulation analysts needs. What this allows is the customization of the simulation engine without resorting to an external language (as is the case with current simulation systems), while still maintaining a high level language interface.

4. NATURAL LANGUAGE PROCESSING ISSUES

The KAMSS architecture presented in Section 3 calls for extensive use of natural language processing techniques. We can break the natural language processing problems in KAMSS down into three basic areas:

1. Natural Language Understanding (NLU),
2. Natural Language Generation (NLG),
3. Discourse Management (DM).

Even though the applications of these techniques in KAMSS are restricted considerably by the modes of operation and the presumed domains, the KAMSS architecture still admits to the most agressive use of this technology proposed to-date. In this section the state of the NLU, NLG, and discourse processing technology is reviewed and methodology innovations are proposed which will provide the basis for achieving the requirements of the design presented.

Another issue to be made in this section is that the analysis required to build the data necessary to construct a natural language interface is also required to design the knowledge representation structures within KAMSS. That is, even if we were not considering a natural language interface to KAMSS, we would need to apply the methodology presented herein to identify the knowledge structures needed to capture and reason about the system descriptions, designs, and models. In retrospect, it may seem obvious that the way one identifies the concepts that people use in their descriptions of a situation is to analyze the language construct they use when they describe those situations. On the other hand, the discovery that the methodology developed could (with minor additions) serve both roles, came as quite a surprise.

4.1 Issues of NLU, NLG, and Discourse Processing

Developing a natural language processing system (NLPS) is an extremely difficult task, although it might at first glance seem deceptively easy because of the ease with which humans, including small children, manipulate and understand language (see [Winograd 1983] or [Sager 1981] for a useful overview of natural language understanding). Natural language understanding (NLU) involves not only understanding the meaning of individual words but also understanding the meaning and function of those words within a sentence or even a body of discourse. However, many English words are ambiguous; that is, the same word can be used either as different parts of speech or with different meanings. Further, in developing an applications-oriented NLPS, we have the classic problems of linguistic theory, including anaphora, elipsis, and coordinate conjunctions, that remain open research questions, as well as the inherent ambiguity of English syntax. Humans process ambiguity because they possess situation, background, or world knowledge that sets the context of the discourse and builds a script of what is expected. It is the difficulty of building into a NLPS this background knowledge component and a method for recognizing the functions of words in blocks of discourse that makes the task of designing computer NLPS so difficult.

Generating natural language is also difficult since it requires not only knowledge of the message to be communicated but also the conventions of the discourse at hand and how to plan the presentation of the message. Natural language generation in the KAMSS is of four basic types:

1. The generation of responses to queries,
2. The generation of summary descriptions,
3. The generation of questions to the user to acquire additional information,
4. The generation of explanations of behavior.

Each of these types of processing has its own associated problems. The generation of responses to queries, for example, is not as simple as the formatting of data base output to the terminal, since many types of queries include analysis requirements which may be beyond the capabilities of the system at any point in time. This presumes the ability of the system to be able to explain its own limitations to the user. The generation of summary descriptions and descriptions of behavior involves the problem of deciding what is meaningful to say, as opposed to saying something about everything the system knows. The generation of queries to the user to clarify information already acquired must incorporate the strategies which an expert systems analyst employs for "leading" and "focusing" a discussion, as well as recognizing when a discussion has reached the limits of knowledge of the individuals involved.

The mode of operation depicted in Section 3 of the user interface throughout the functionality of KAMSS was one of an interactive discourse. This mode of operation, and the fact that some of this discourse involves making reference to information in picture form, requires the characterization of each discourse situation. By characterization of the discourse situation we mean at least a basic model of both "partners" in the dialog. At least one model for each type of user during each mode of use of KAMSS is required. The theory of semantics presented in Section 5 provides the basis for such a characterization, and the reasoning method outlined in Section 6 provides the means for using this representation to accomplish the NLP tasks associated with the KAMSS. One of the tools developed as a part of this reseach (IDEF-1/ES) has the capabilities of representing these types of discourse models.

A NLPS conceptually consists of a parser, a semantic interpreter, and optionally a text generator. The parser determines the syntactic structure of an utterance while the interpreter assigns meanings to these structures. To assign words to syntactic categories or parts of speech, the parser makes reference to a lexicon as discussed below. Once assigned a part of speech, these lexical items are then assigned to larger syntactic structures such as noun phrases, verb phrases, or prepositional phrases according to a grammar component. The semantic interpreter is responsible for the construction of a representation of the meaning of the utterances processed by the parser. The ability of this component to adequately perform is dependent on the underlying semantic theory which will be discussed in detail in Section 5. The text generator operates in essentially the reverse order. It must be handed a strategy which encompasses the message to be communicated and the raw data on which to operate. It will then formulate a discourse framework for communication of the message. This framework must then be converted into natural language sentences. These sentences must then be refined into an acceptable text structure and stylization performed to eliminate redundancy and awkward wording. Finally, the resulting text must be displayed to the user.

In general, semantic interpreters work by combining word meaning (generally extracted from the lexicon or an associated dictionary), sentence structure, and discourse scripts to classify utterances and thereby produce a "meaningful" representation. The adequacy of the classification is evaluated either by examination of the resulting actions which are taken upon input, or by the ability to answer questions relative to a particular input. In the KAMSS as opposed to other natural language processing domains, the text generator is a major component of the NLPS. One of the major skills that an analyst brings to the problem is the abilty to provide summarizations, critiques, explanations, etc., on the system descriptions which have been provided to him. We view the text generation problem as beginning after the reasoning or other data processing activities of the KAMSS. Thus the function of the text generator is to take a coded form of a message and produce an acceptable English language statement — essentially the reverse of the parsing stage.

Central to each of the NLPS activities described above is the lexicon. In its simplest form, the lexicon is a dictionary of possible lexical items and their associated syntactic categories. However, such a simple lexicon has a serious drawback: many English words are inherently ambiguous on several different levels. Firstly, a word may be syntactically ambiguous, with several possible category assignments. Thus the word *run* may be either a noun or verb; the word *that* may be a pronoun, determiner, or subordinator. Secondly, a word may be lexically ambiguous, with several different meanings associated with the same syntactic category, as in the word *run* which can be used as either a verb or a noun. As a verb, it has a range of meanings from "*run* a race" to "*run* away" to "*run* a risk" to "*run* a fever", while as a noun it can have such various meanings as in "a 10K *run*" "a long *run*" "the usual *run* of men", or "a *run* in my stockings". Humans easily process and understand these various senses because of the semantic cues provided by the larger body of discourse. Unfortunately, a large number of English words are ambiguous in one or both ways.

4.2 Processing of Utterances

Under the design presented in Section 3, each major mode of operation of the KAMSS (description capture, model generation, description and model analysis) allows for the input of three types of unstructured "utterances" (text, discourse, and sketches). The text input serves to support the function of entering directly company policy and procedure documents. Text input processing is also necessary to support the use of past session discourse as context for a current discourse. As described in Section 3, one mode of information storage which is required is that of annotated text. The discourse input serves to provide the interactive processing of statements, commands, declarations, and questions. Discourse generation serves the function of supporting the formulation and presentation of responses to these inputs. Thus the required KAMSS features of explanation, commentary, data base interface, and report generation are supported by the discourse generation capabilities.

4.3 Overview of Existing NLP Methods

The following subsections provide an overview of the state of the technology and the issues associated with the understanding of natural language utterances (NLU), and the generation of responses formulated in natural language (NLG). These issues are presented for comparison and background to the approach recommended in the following sections.

4.3.1 NLU Approaches

The best strategy for actually parsing and understanding sentence components of text is an open research question. Of the many prevailing approaches to NLU, most fall into two broad categories — linguistic or conceptual. ([Marcus 1980] is an example of a strictly linguistic based approach while [Schank 1982] is an example of the conceptual approach.) Linguistic systems maintain the parser and interpreter as separate components, but conceptual systems use some elements of the interpreter to constrain the parser. One such constraint mechanism is the use of case frames [Fillmore 1968], that is, the assignment of syntactic categories based on the semantic / syntactic relationship of constituents to the predicate of the sentence. Another, is the use of a sublanguage system [Sager 1981], or a lexicon constrained by a particular domain.

For instance, each lexical item in the sentence:

The shop has experienced MBA trained supervisors.

represents a single syntactic category except for the word *experienced,* which has two syntactic categories. The interpreter of a linguistic parser would have to decide between these two interpretations, one of which would take the word *experienced* as the past participle of a verb synonymous with *suffer.* The second would assign *experience* to the adjective category, with the meaning "*made capable by reasonable experience.*" A domain specific conceptual parser might eliminate the second reading altogether by assuming that in the manufacturing domain *experience* will only modify human nouns when it is used as an adjective. Such a semantic constraint would be imposed as part of the information given in the lexicon, either about the syntactic category of *experience* or about the case roles that it may fulfill. Conceptual parsing strategies differ in how lenient these constraints are.

Linguistically based systems rely primarily on knowledge of grammar (the syntax and morphology of a particular language) rather than on knowledge of a particular domain. Because these systems are syntax-driven, they may actually generate several meanings for the same sentence and may in fact generate interpretations that make no sense in the real world. Conceptual parsers, on the other hand, are guided in their parsing by their knowledge of some domain and thus eliminate interpretations which have no meaning for a particular domain. However, since conceptual systems are domain-specific, they cannot easily be generalized without the regeneration of the knowledge component. The grammar in a linguistic parser usually consists of a set of phrase structure rules used to bundle syntactic categories into syntactic constituents, but in a conceptual parser the grammar forms a set of sentence patterns anticipated by the system. Conceptual parsers attempt to incorporate some of the human's ability to disambiguate by providing extensive knowledge about a limited domain, thus imposing severe constraints on the use and co-occurrence of certain lexical items. For instance, in a conceptual parser, the verb *see* would require an animate agent and an optional indicator of manner. These constraints are imposed by syntactic category restrictions on words in the lexicon and by case frames associated with verbs.

Regardless of the approach, parsers accept as input a subset of the text (we will assume a sentence by sentence parse) and match each lexical item (or word) with possible syntactic categories (or parts of speech). The string of syntactic categories are bundled into larger and larger syntactic constituents by the parser until the sentence can be processed, and the information content of the sentence incorporated into some representation strategy.

4.3.2 NLG Approaches

There are four basic issues associated with the generation of natural language responses. The first is deciding "what" to say. The second is deciding "how" to say it. The third is the actual formulation of the textual response (i.e., the speech act). And the fourth is deciding how to integrate the generation process with the understanding process and the "flow" of the discourse. As with the NLP, the syntactic portions of these issues (the second and the third issue) have been more extensively studied than the others. This is presumably because of the fact that the syntactic issues lend themselves to the application of more general methods.

One common class of language generators which are quite extensively used are those which generate "canned" messages. For example, in a language compiler certain "canned" messages are produced when a particular error is detected. In some cases these "canned" message generators can be quite sophisticated, allowing the specification of variables in the message that can be bound at text generation time to produce reasonably tailored messages which are situationally accurate. The use of these methods should not be discounted; in fact, there are many places within the KAMSS where such techniques will be the approach of choice. Nor, should one discount the convincing nature of such approaches relative to the perception of the user that he is communicating effectively with the machine (any time spent working with the ELIZA system [Weizenbaum 1966] will probably convince the user otherwise!) The problem with such approaches is the extensive customized programming which is required to implement them, and the fact that they are relatively inflexible and suitable only for short statement generation. On the other hand, they tend to be computationally inexpensive.

Early language generators merely produced random syntactically correct sentences based on the transformational models [Yngve 1962; Friedman 1966]. The first attempt to map semantic structures into the syntactic forms was built upon the use of augmented transition networks (ATN) [Simmons and Slocum 1972]. ATNs were originally designed by Woods as a computational technique for the processing of text [Woods 1970]. Simmons essentially turned the process around so that the arcs in the net were labeled with semantic "case like" components rather than syntactic ones. The basic problem with the use of ATNs is that the computational paradigm was too powerful. In other words, their use was roughly equivalent to the production of a custom tailored program with the semantics hard-coded in. Goldman's BABEL was the first serious attempt to separate the semantic functions from the final text generation [Schank 1975]. BABEL used a discrimination net representation of a set of semantic interpretation rules to generate a "case frame" like structure which was then translated into the output text via an ATN based system. By variations in the semantic interpretation rule preferences, the verb and case frame decisions made by the semantic component could produce paraphrases on equivalent input.

Two programs which established some significant capability in discourse generation planning were PROTEUS [Davey 1978], and NLPS [Heidorn 1972]. Both of these systems provided capabilities for analyzing structured domain representations and determining what linguistic structures would be suitable for describing (summarization in both cases) the information in those representations. Both systems relied on a fair amount of domain specific knowledge (the NLPS domain knowledge was described earlier in Section 1, PROTEUS's domain knowledge was centered on the domain of the "tic-tac-toe" game). The PROTEUS system examined the move record and attempted to identify "contrasts" and "intentions" as signified by the actions in that record. PROTEUS also attempted to eliminate uninformative text by assumptions made relative to the fact that the person receiving the explanation was the person who had just played the game. Thus it demonstrated (in a hardwired form) the benefits of using the discourse situation as a frame of reference for text planning. NLPS generated questions concerning the incompleteness of a specification primarily by the examination of required slots in a frame representation (e.g., moving objects have an arrival distribution, etc.). Summary generation was driven by a set traversal of the records within the frame.

The use of more general planning methods for text planning was first explored in the Knowledge and Modalities Planner (KAMP) [Appelt 1980]. This system used hierarchical planning methods [Sacerdoti 1977] to generate a gross plan to achieve a goal in a robot planning domain. This gross plan (set of actions) was then verified using the "possible worlds logic" of [Moore 1980]. Once the gross plan was verified the system examined each action to see if it was primitive and if not, recursively applied the planning procedure to that action. The resulting task plan was then syntactically processed into a grammatically correct sentence. In the strategic planning for the discourse KAMP addressed the problem of using the systems knowledge of both the situation and the facts that the system could assume the user to know about the situation. For example, if a component was known to be attached to only one other component KAMP knows that it is sufficient to merely request that the component be removed since it is "obvious" what it is to be removed from. KAMP also knew that if it told the user to use something then it must also tell the user where that thing is (e.g., use the wrench *in the tool box*).

4.4 Methodology for Utterance Analysis

From an engineering point of view, the central issues of NLP and NLG reside in the question of how to build usable implementations of these capabilities for a particular application domain. All of the technical papers, reports, and texts dealing with the issues of NLP and NLG focus exclusively to the general issues of language phenomena (syntactic or semantic), language use, or computational methods. Therefore, one of the technical engineering voids in these areas was the general lack of a "usable" methodology for applying these results to a particular problem domain. This phenomena itself is interesting since in most engineering domains the practice or methodologies normally proceed the theory or method. In this section we describe a four step methodology for performing the analysis necessary to collect the data for, and to design usable NLP and NLG systems.

Methodology Assumptions

The primary assumptions which are evident in the methodological approach which we have used is that:

1. Communication via language is an environmental phenomena,
2. Actual use of language is inherently constrained by the processing capabilities and limitations of the human being,
3. Cognitive activities heavily influence the generation and processing of language during a communicative act,

4. Relative to the formulation / discovery of the underlying concept structure:

    4.1. Most reasoning processes are dominated by a matching process which is largely syntactic in nature, resorting to semantic considerations in the minority of cases [Bobrow and Winograd 1977],

    4.2. Understanding can be effectively modeled as theorem-proving in a constrained specific situation [Hobbs 1986],

    4.3. Classification structures are useful in the reasoning process, where as the underlying general principles are the key to understanding, thus both are needed in a NLU / NLG system,

    4.4. Deep knowledge comes, not from the selection of a single ontology, but rather from a rich set of alternatives which have been cross-correlated.

Based on these assumptions the following methodology will be seen to be quite "bottom up" or data driven. It can actually be characterized as a knowledge engineering type activity in that it is based on a belief that to understand language produced in a particular domain, one must study samples of that language as produced by participants in that domain.

The first step in the development of a natural language processing or generation system is the assimilation of a corpus of examples from the domain of discourse where the system would be expected to perform. From experience, it would appear that the domain specific patterns which must be identified tend to become evident from as few as five different discourses particularily if they were generated by different speakers. In a study of patient medical history records over 100 samples were processed [Mayer et al. 1987]. However in this dissertation work we limited our analysis to 15 samples. Such a limitation was partially justified by the length of the examples analyzed (i.e., each manufacturing example was 3 to 5 times longer than the typical medical text.) Another problem encountered in this research was that manufacturing system descriptions are not generally recorded. Therefore, we had to rely on personal interviews with manufacturing experts to augment our analysis.

The second step in the process is the analysis of the individual sentences in each text using a systematic grammar classifying the following types of information:

1. The types of utterances (text, discourse, lists, sketches) and the function(s), form, and structure of each type of utterance,

2. The types of sentences (indicative, declarative, demonstrative, interrogatory, commands, interjections) the function(s), and form (simple, conjunctive, conditional, etc.) of each type of sentence,

3. The types of clauses that make up sentences (relative, subordinate) and their function(s), form, and fit within the utterance,

4. The types of phrases that make up clauses (subject, adjectival, adverbial, etc.) and their function(s) and form,

5. The lexicon (the set of words) used in the text and what are their associated function(s), form, and fit.

A useful set of structures for this analysis is depicted in Figure 4.1.

The third step in the analysis process is the production of semantic models of the concepts commonly used in the particular domain of discourse. The lack of an effective semantic modeling method for use in this analysis prompted the development of the IDEF1/ES described in detail in Section 8. An example representation of semantic information represented in this methodology is presented in Figure 4.2. The process for this step can be characterized by the following four steps based on our experience and that reported in [Hobbs 1985, 1986]:

1. Concept identification and characterization. This step requires examination of the individual words and phrases in the text and examination of uses of synonyms, metaphors, and classification schemes. As concepts tend to cluster [Hayes 1979], grouping related phenomena references together is often a good starting point for the concept identification. Once the concept set is established, it is useful to attempt to separate the set into primitive versus composite concepts. A concept characterization should focus on determination of as large a possible set of necessary and sufficient conditions for a concept without being overly concerned with finding them all.

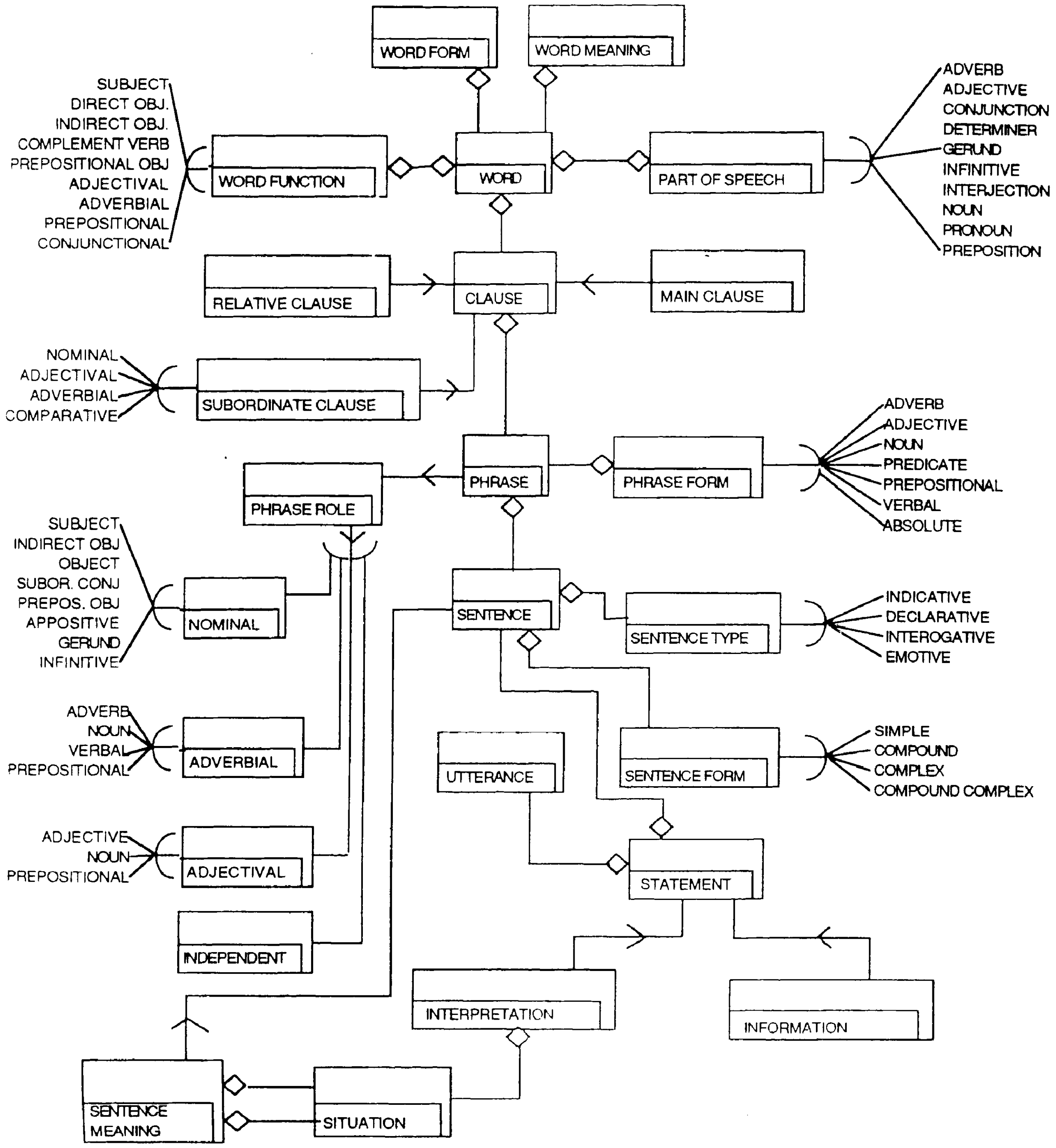


FIGURE 4.1: CLASSES FOR UTTERANCE ANALYSIS.

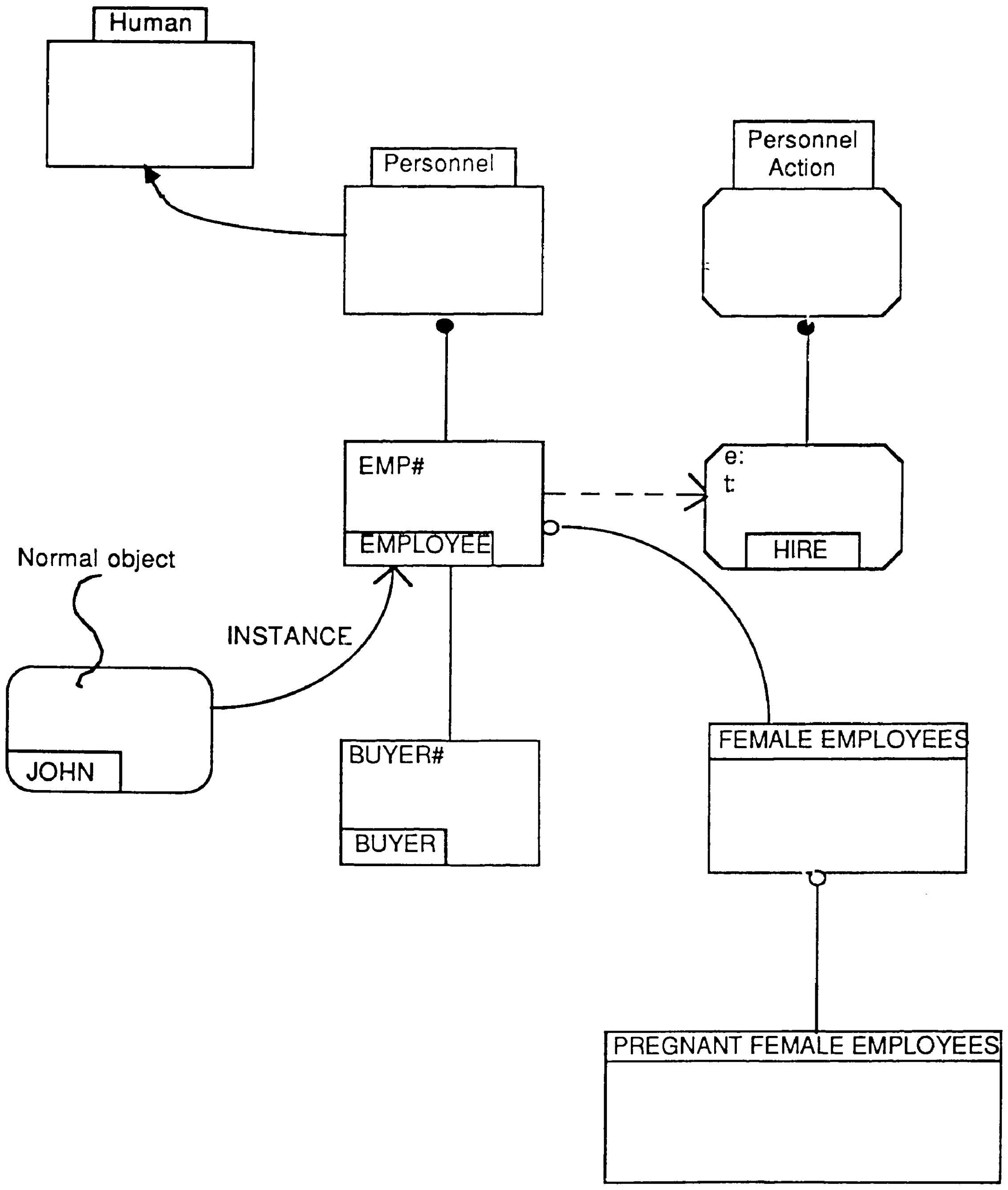


FIGURE 4.2: IDEF1/ES SEMANTIC MODEL.

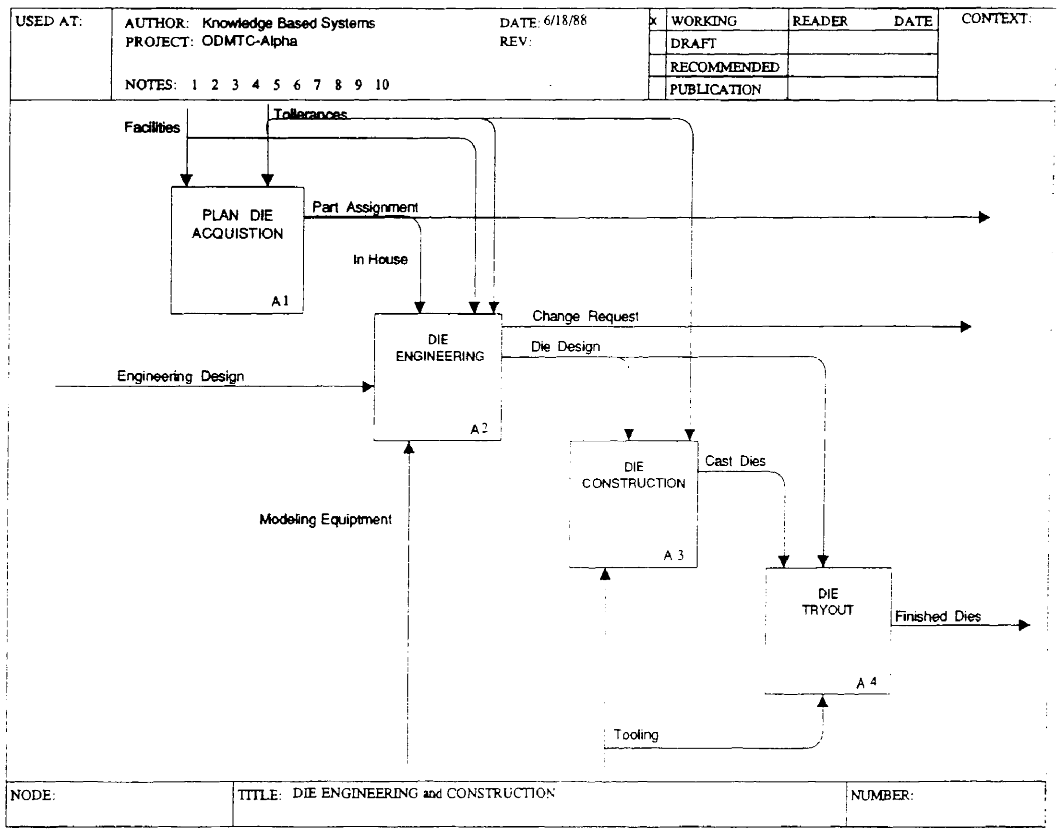
2. From the text, construct as many axioms as possible introducing new concepts freely as required. The process of axiom construction from the system description is one of the most powerful means of concept discovery.

3. Determine the minimal structure required for a concept to be usefull in relation to the other concepts. This allows for a concept to be reused in other contexts without major modifications.

4. In the analysis for redundancy in concepts rather than focus on selection of a single concept where duplicates occur use that as a possible indicator of overlapping ontologies. Attempt to formulate each and show how the primitive set of one can be characterized in terms of the primitive set of the others.

The fourth step in the analysis for construction of an NLP or NLG is the identification of the basic tasks which the planning portion of the processor or the generator must perform. These tasks can initially be identified by the construction of IDEF0 models of the domain [IDEF0 1980]. Past experience with the analysis of IDEF0 models constructed by domain experts with no experience in the technology of artificial intelligence has shown that they accurately characterize the thought processes of the experts, and can reliably be used as a basis for the design of knowledge based planning systems [Mayer 1985; Friel and Mayer 1985]. An example of part of an IDEF0 model constructed for a machine fault diagnosis expert system is presented in Figure 4.3 as an illustration of the use of this method.

Using the above four step analysis process will produce the raw data necessary to build a NLP or an NLG. The actual manipulation required on this data for production of the system depends upon the tools chosen. If the tool is a case frame based system such as Language Craft [Carnegie Group 1986], then the "relation labels" off of the IDEF1/ES model map directly into sentential case frame concepts. The "entity classes" map directly into nominal case frames, and the "types" map directly into schema structures.



USED AT:	AUTHOR: Knowledge Based Systems PROJECT: ODMTC-Alpha NOTES: 1 2 3 4 5 6 7 8 9 10	DATE: 6/18/88 REV:	x	WORKING	READER	DATE	CONTEXT:
				DRAFT			
				RECOMMENDED			
				PUBLICATION			

NODE:	TITLE: DIE ENGINEERING and CONSTRUCTION	NUMBER:

FIGURE 4.3: USE OF IDEF0 IN DOMAIN ANALYSIS.

4.5 Approaches Recommended

The following sections describe the approaches to NLU, NLG, and DM which have been developed to serve the requirements of the KAMSS architecture presented in Section 3.

4.5.1 Natural Language Processing Approach

In the course of this research, we have experimented with several different strategies for understanding natural language. For command processing, we have adapted the concept type driven template completion approach. For rule / axiom capture, we used a combination of the case frame and type matching approaches. Under this approach the user interactively constructs his statements within an interactive editor. At each stage of the statement formulation process the interactive parser controls the choice of case frame templates which the user can apply. The semantic interpreter does interactive checking of the semantic categories of the input, signaling errors when there is an explicit type conflict, and performing type coersion when an unclassified description is encountered. Thus, all of our parsing and understanding is designed to be interactive where inconsistencies and ambiguities are avoided by the structures provided or eliminated by interaction with the user. We have also experimented with processing of large texts for specific types of information content using a multiple pass parse / interpretation, as described in the following paragraphs.[1]

[1] My initial experience in this area was with analysis of the problem of automatic processing of clinical medical record texts in a research program with Scott and White Medical Clinic. Dr. Guy Bailey served as a Co-Principle investigator on the initial effort along with myself Paula S. D. Mayer and Dr. John Dvorcek, and Dr. Argie Hillis. Our initial attempts to use a strict case frame approach failed misserably for reasons described in detail in [Mayer 88]. Paula developed a multipass conceptual approach which proved successful not only for the medical application but also for limited tests in the business rule and manufacturing description processing area. However, as will be discussed later in this section while this approach does appear to offer substatial promice for KAMSS applications it cannot handle the complex conditional logic descriptions which were quite common in the manufacturing texts examined.

A conceptual approach to NLU is suggested because of our belief that successful and efficient interactive discourse processing and understanding depends upon building a computer system that combines grammatical knowledge with the expectations and constraints that an analyst brings to the processing of customer input. Thus, we reject the notion (for the KAMSS application) that a parser can operate autonomously on syntax, without regard to the domain of discourse. The prototype parsers constructed by Paula Mayer [Mayer et al. 86 and 87] during the course of this research used expectations of textual structure built into patterns based on the information structure of the material, the concept ontology itself, as well as grammatical expectations or case frames. For these reasons, the systems would not understand texts which lie outside that domain, such as newspaper articles or even articles in manufacturing journals. The following paragraphs describe the basic workings of a conceptual approach to NLU for KAMSS followed by a comment on the actual potential of such technology for application in the architecture presented in Section 3.[2]

The patterns (required by such a system) for system description texts would expect information relating to a facility or organization, either reported by the customer, or observed by the analyst. The patterns also anticipate either the verb and preposition or the verb and case frame cooccurrence structures that help establish case frames. Since these structures are complex, the parser must sometimes rely on a scale of probability for understanding the meanings of lexical items which have a wide range of meanings. The parser, for instance, would attempt to understand *run* as referring to a process (e.g., to *run* a job) or perhaps a state (e.g., a *run* on a part); it would not, however, understand "*run* in a stocking" or other meanings of *run* outside of the manufacturing domain. The current strategy also makes use of domain specific constructs which border on "sublanguage" phenomena. For example, in manufacturing one almost never uses a phrase such as "The operation inspection" though that would be grammatically correct. Rather one normally refers to "The inspection operation". Taking

[2] For a detailed description of the representation strategies and reasoning strategies of this conceptual approach and its application to clinical medical text processing see [Mayer 88].

advantage of these domain constraints assists in both the parsing (described in the following section) and the semantic interpretation (described in Section 5).

One of the ways to constrain the complexity of the text processing problem is to isolate different modes of access / use of the system. In the domain of manufacturing description texts, for example, we structured our set of patterns to expect text in the first or third person. For instance, we certainly do not expect imperative intermingled with indicative sentences. In fact, we use canned text generation to deal with imperatives (e.g. "Sorry, you can't do that to a computer"). Moreover, most of our patterns anticipate that text in the third person refers to the system. Further, these patterns are associated with processing sentence frames which anticipate certain types of information once the verb phrase is isolated. Since this information is often denoted by the co-occurrence of verbs and prepositions or verbs and syntactic functions (such as subject or direct object), our processor is tuned to search for the anticipated semantic casemarkers.

Because of their importance in system descriptions, an NLPS must deal with time, place, movement, symptoms, system elements, attributes of system elements, their relationships, and states. In the development of our prototype, we have concentrated on formulating the basic concepts necessary for capturing and making inferences about the objects, their relations, states, and activities. We have tried to develop these concepts independently of any single parsing strategy. However, our current parsing strategy is based on a control mechanism as follows. First, a gross parse of the sentence is made in order to get the essence of its information content and to break complex sentences up into elementary components. This is accomplished through the use of a heuristic of trying to isolate the verbs of the sentence and also looking for special composition structures (e.g., *if...then, subordinate or relative clause markers, etc.*). Each sentence fragment identified is sent to be processed by an element of the grammar database and lexicon dealing with the particular information content associated with that verb in a second pass of the parser (performed by Language Craft). It should also be noted that this strategy is heuristically based; that is, it is not immune to giving false results. It may, in fact, misidentify a word that is ambigious. The robustness of the heuristics is based on the completeness of the co-occurrence sets and patterns defined. These sets and patterns have been derived from linguistic analysis on actual manufacturing system description texts.

In the second parsing stage, the processor further parses and builds a data structure of the information content into an appropriate case frame. This is accomplished by further analyzing those sentences based on the type of verb or noun phrase that occurs. For example, verbs which carry locative information can be grouped into a small set of classes based on the case frames with which they are used, thus providing the parser with a limited set of patterns. We have found four groups of verbs that either contain or imply locative information in our corpus of manufacturing texts. These verb classes are shown in Figure 4.4. These verbs differ in the type of locative and temporal case frame markers that follow. The casemarkers, generally prepositions, are grouped into classes based on two criteria. First, the locative and temporal casemarkers are in separate groupings. Secondly, the casemarkers — whether locative or temporal — are also grouped into classes based upon the syntactic patterns in which they occur. Figure 4.5 shows a hierarchy of the casemarker's groups.

In the prototype system, the processing of all utterances uses the notion of a minimal lexicon. This approach was dictated by two considerations. First of all, human analysts do not operate with a complete lexicon; rather they have a fairly well developed capacity for inferring lexical semantic categories from contextual cues or simply by asking. Second, it is not practical to try to preload the lexicon since this would require a priori doing the work that the KAMSS system description capture enviroment was conceived to do. For example, we cannot incorporate every geographic place name (state, county, city, country, area, etc.) in the world in the lexicon; further, some places may be indicated by directional indicators (e.g., "*west of here*") rather than by specific place names. Additionally in the domain of system description capture, as pointed out in Section 2, a considerable amount of the human effort which goes on in problem analysis and solution design activities is the creation of names (lexical entries).

Thus, we propose the use of a minimal lexicon with usage patterns and restricted case frame structures for accomplishing a flexible text processor. For example, part of the system based on characteristic geographic patterns derived from our linguistic analysis of the manufacturing texts. Locative information may be present as either the object of prepositions in verb / preposition co-occurrence sets or in the various syntactic functions associated with verbs, but both rely on successfully isolating the verb. Once the verb is isolated, the system will detect

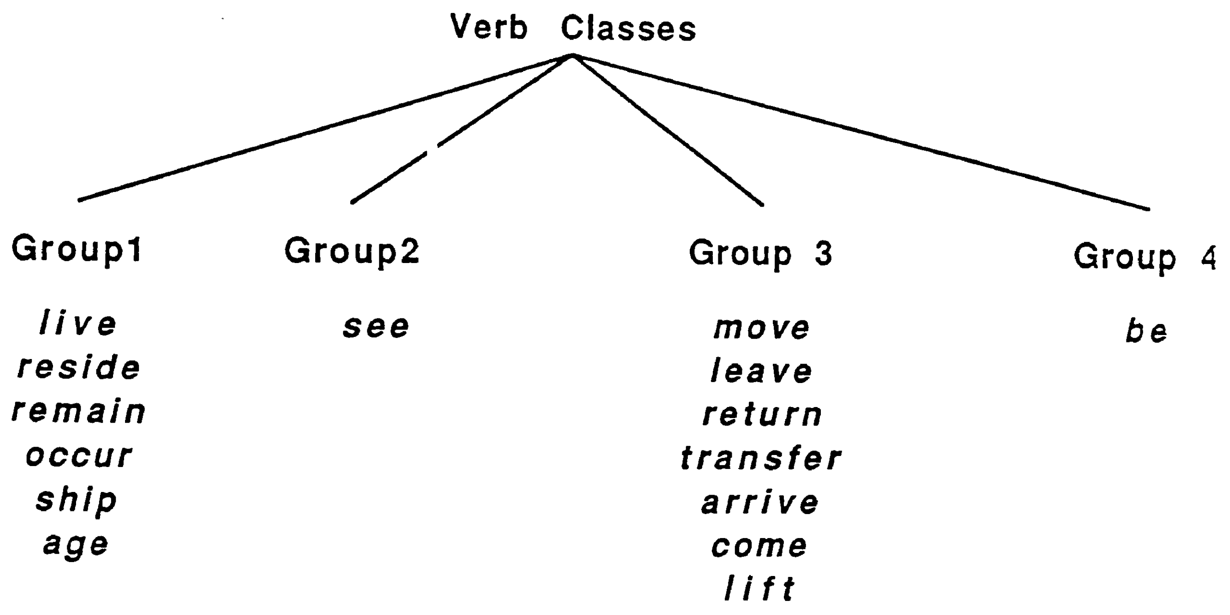


FIGURE 4.4: VERB CLASSES IN PROTOTYPE KAMSS.

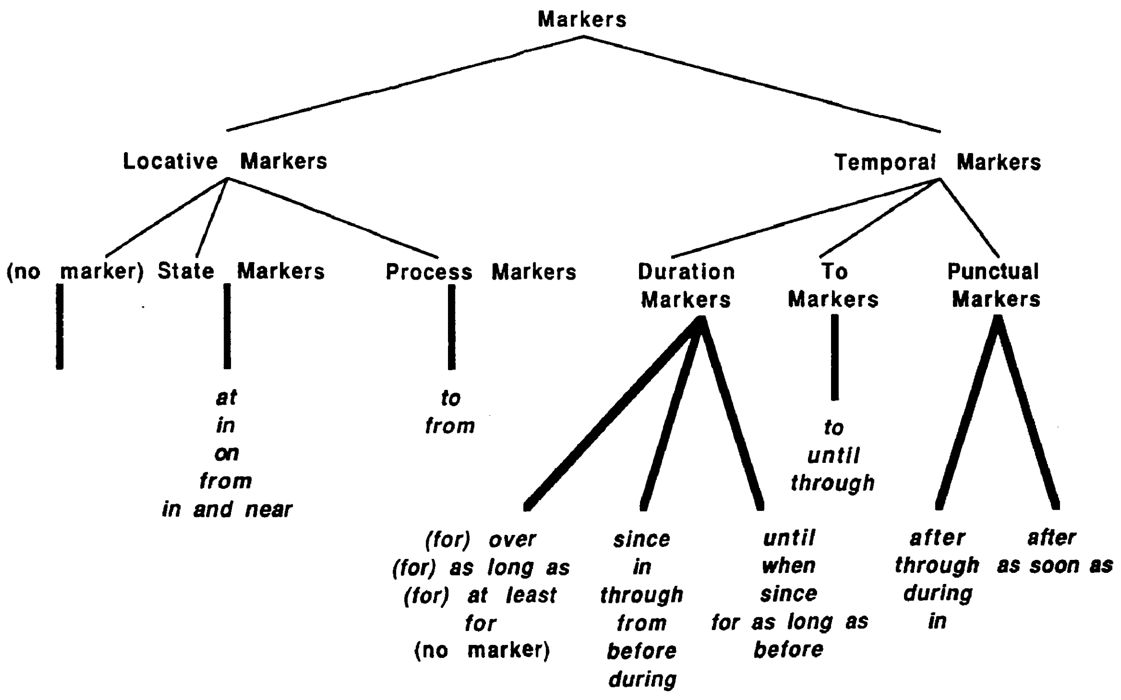


FIGURE 4.5: CASEMARKERS AND THEIR HIERARCHY IN KAMSS.

geographic locative information in one of two ways: by actually matching the co-occurrence sets in the case of locative information marked by designators or by eliminating all other possibilities. This elimination of all other possibilities is used in identifying proper nouns of geographic locales. For instance, if a phrase is determined to be locative but does not fit a pattern, it is assumed to be a proper place name. As a result, we have only the smallest lexicon necessary for isolating sentences with information about place and time.

The conclusions which we can reach from our experimentation and interaction with the potential end users of a KAMSS are as follows:

1. Interactive expectation parsing and understanding are sufficient for most of the description acquisition, query processing, and command interface tasks. This approach not only minimizes the input requirements of the user but also provides interactive guidance to the user enabling a tutor like support in all modes of system use.

2. The fact that our investigation uncovered few written textual descriptions of manufacturing systems we feel indicates:

    2.1. That creations of even skeleton system descriptions in a purely text mode is difficult because of the complexity and quantity of the information which must be provided and because of the lack of time and authoring experience of those personnel with the knowledge to create such descriptions.

    2.2. The need for a powerful unrestricted natural language text understanding system is minimal.

3. There is evidence to the fact that the "copy and edit" form of description acquisition (as well as knowledge acquisition) is highly efficient. We feel that the most promising path to pursue in the future is the extension of the expectation based parsing to cover entire textual passages. That is, we would like to present to the user textual descriptions of various sizes which have been preprocessed (i.e., the parsing and interpretation have already been done) the user could use these descriptions as a starting point. The copying, editing, and submission of such texts as system descriptions would then give

the impression of unrestricted language understanding at an acceptable speed and with minimal typing input.

4.5.2 Natural Language Generation Approach

The extent of the requirements for natural language generation (NLG) in the KAMSS dictates an approach which cleanly decouples the domain knowledge from the generic "language competence" in the generation process. To this end, we recognize that there are actually three levels of NLG planning. The first level is strategic planning which is responsible for the definition of the information content which will be conveyed. The second level is the situational level which is responsible for determining how the context of the current discourse can be manipulated to augment or reduce the information to be conveyed. The third level is a syntax planning level which is responsible for the determination of what syntax and lexical options exist for communication of the required information. Underneath these planning levels is a "competence" level which is the actual utterance generator.

By making these levels distinct, and by having an underlying "competence" level which is responsible for the actual production of the utterances of as per the output of the planning levels we can achieve a structuring of the three types of domain knowledge necessary for a flexible NLG capability. Thus, for example in the MDS subsystem, the same strategic level planning rules for the generation of the English description of a model could be used for the generation of the actual code. The diffence between the English output and the simulation language output would be accommodated in the syntax planning level and the "competence" level.

The different levels of planning draw on different knowledge sources for the planning at that level. For example the competence level draws on syntactic knowledge of the morphology type for word generation and on lexicology rules for handling such issues as number and tense agreement. The syntax planning level draws upon the case frame structures used in the understanding component to determine suitable ways of structuring information presented to it by the strategic and situational planning levels. The strategic level draws on general knowledge about:

1. How to summarize a system description.

2. How to ask a question.

3. How to answer a question.

4. How to describe a situation.

This knowledge is deeper than it might appear. For example, summarizing a system description requires having a general strategy for deciding what is within the context of the system. It also requires a strategy for describing what something is (i.e., presentation of a physical objects attributes and its relationship to other physical objects). Finally it requires knowing how to describe how something works (i.e. describing the goals, the role or function that each component plays to cause or effect the actions which achieve those goals). This includes common sense strategies such as "when a device is operating the way it should it is common to associate the purpose of an event or action with the results of that action. The strategy generator must also be sensitive to the representation of conditional constraints (explained in more detail in Section 5). As pointed out in [Forbus 1984] constraints are inherently non-causal in nature. Just knowing that $F = ma$, for example, does not allow us to explain an increase in mass as an increase in force with a corresponding decrease in accelleration. Causal explanations between essentially independent variables in a constraint should be avoided by the strategy generator.

Strategies for answering questions can vary from tabular or graphic display of data directly from the description base to complex explanation of the cause / effect relations. Relations, for example, that give rise to an observed failing in a system (in attempting to answer a "why doesn't it work" question), or the generation of a description of a possible world situation in which a proposed system would fail (in attempting to answer a "will it work?" question).

The situation level can be thought of as a filter between the strategic and syntax planning levels. This level takes into account the resource situation provided by previous discourse to reduce the amount of redundant information presented to the user. This level also includes rules which govern textual and stylistic constraints (e.g. avoiding redundant use of words when alternatives exist, variations in the order of the clauses produced, or over/redundant specification of entities). Some of this kind of processing is also provided by the strategic level in that the knowledge about how to generate text contains information on how to use cohesive structures such as pronouns or other forms of anaphoric reference to refer back to previously presented information.

As a part of this research we have developed two text generation systems. The first works off of a model (specifically an IDEF1 information model) and constructs an English text description of the assertions which that model makes about the world it purportedly represents. The second constructs a textual summarization of a system description based on an instance specification based on an ontology in the description knowledge base. The conclusions we can make based on the results of our research development regarding text generation are as follows:

1. Generation of acceptable natural language output is highly context dependent.

2. The more clearly defined the context for text generation the easier the task of producing acceptable text.

4.5.3 Discourse Management Approach

The discourse management approach is directly influenced by the underlying KAMSS reasoning mechanism and the overall system structure. The kernel structure within KAMSS is based around the concept of an analyst in a box. That is, the user is expecting the system to react to his input the way an expert systems analyst would interact with him. This leads to the following requirements on the discourse manager:

1. The discourse manager must be aware of the passage of time:

   1.1. Between the receipt of a piece of information and the response (if any) to that information,

   1.2. Between the issuance of a request or query of the user and a response,

   1.3. Since the beginning of the session, 1.4. Since a request for a particular service.

2. The discourse manager must be able to focus and direct the interaction during all modes (description acquisition, analysis and modeling),

3. The discourse manager must be able to support the issuance and management of multiple services during all modes.

Requirements 1, and 3 above are primarily monitoring in nature and can be addressed with traditional operating system techniques. Therefore we will focus our attention on Requirement 2. We propose addressing the focusing of an interaction by providing the discourse manager with access to a model of a typical usage scenario for the task at hand and the ability to determine when that model is being tracked. In analysis of actual user interactions with analysts, it is often difficult for the human analyst to determine when the user is wandering in the discourse. There are many different kinds of focus shift. Some of these are natural discourse mechanisms for human to human communications. For example, often a user will elaborate in detail past situations or events (war stories) in order to emphasize a relation which he believes is relevant. We believe that such problems, which arise in interactive story telling, will tend to be rare if the proper description acquisition framework is provided for accepting the "axioms" inherent in the experience communicated verbally by the "war story".

There are indicators which can be triggered upon. For example, repeated reference to "the problem" or phrases of the sort "what we need is" typically are used to point blame or describe solutions. Use of phrases such as "requirements of" and "requirements for" again generally indicate change of focus from the current system description to description of a proposed solution. Repeated reference to specific persons or use of past or future tenses of verbs can also be indicators of focus shift. We also propose providing a concept database which can be programmed to contain "concepts" which typically indicate focus problems (references to "family," etc.).

We also will be able to use the "manufacturing" knowledge of the system and the user profile (title and functional responsibilities) to help identify change of focus. As we are proposing a knowledge representation scheme which is indexible by viewpoint (see Section 3) we can detect reference to objects which are "outside" of that viewpoint. These references cannot be off hand discarded since they may be essential components in the users understanding of the interface of his domain to another. The KAMSS (at least in the description capture mode) will be attempting to build a representation of the "system" being described. If a large number of disjoint objects are described with no relation between them established, then the system can assume that there is a focus problem with the input it has received. What can be done about this is to periodically have the system attempt to generate a "scenario" which delineates a "flow" through the system description focusing on situation entailment. If we represent each situation in that network as a node, then a crude (but reasonably effective) strategy is to ask about the disconnected components of that network.

The conclusions we can make based on the results of our research development regarding discourse management are as follows:

1. The strategy employed must be flexible, allowing extension by the maintenance user for a specific site situation.

2. The focusing problem is by far the most difficult in theory, however, in practice the use of scripts of text sentences and hypertext annotation facilities with a human analyst support facility can create an effective substitute.

4.6 Summary

In this section we reviewed the of natural language processing technology called for in the KAMSS architecture presented in Section 3. We overviewed the natural language processing problems for KAMSS in three basic areas:

1. Natural Language Understanding (NLU),

2. Natural Language Generation (NLG),

3. Discourse Management (DM).

Even though the applications of these techniques in KAMSS are restricted considerably by the modes of operation and the presumed domains, the KAMSS architecture still admits to the most agressive use of this technology proposed to-date.

An important concept presented in this section was that the analysis required to build the data necessary to construct a natural language interface is also required to design the knowledge representation structures within KAMSS. That is, even if we were not considering a natural language interface to KAMSS, we would need to apply the methodology presented herein to identify the knowledge structures needed to capture and reason about the system descriptions, designs, and models. In retrospect, it may seem obvious that the way one idėntifies the concepts that people use in their descriptions of a situation is to analyze the language construct they use when they describe those situations. On the other hand, the discovery that the methodology developed could (with minor additions) serve both roles, came as quite a surprise. We presented a methodology for performing this analysis. In Section 8 we will present two modeling methods which can be used to augment the described analysis methodology.

5. ONTOLOGY AND REPRESENTATION STRUCTURES FOR KAMSS

Section 2 described the overall cognitive processes involved in decision making using models and simulation analysis. Section 3 described an architecture for a knowledge based system to support this process based on the concept of mimicking the capabilities of the analyst. Section 4 described the techniques required to support the processing (acquisition and generation) of the syntactic form of inputs to (and outputs from) such a system. The first part of this section investigates the important issue of how the system is to be aware of the "meaning" of the symbols it is acquiring. The methodology for semantic theory development is presented in the form of a small set of concepts and a collection of operators for construction of well-formed conceptual structures which is intended to provide a needed generative theory of the structure and limits of meaning acquisition and use. The second part of this section describes a set of the key ontological issues raised by the various knowledge representation and reasoning problems associated with the KAMSS. We particularly focus on the problem of representing descriptions of both the products and the systems in an engineering or manufacturing environment. We then show how these issues can be addressed with the conceptual structures presented.

An important concept that is introduced is the nature of a description as a collection of facts. Our experimentation with attempting to capture system descriptions, object descriptions, design descriptions, and context descriptions over the course of this research convinced us that the available representation schemes were inadequate for any of these tasks. The schemes available to date force descriptions to be constructed as instantiations of prototypical structures. Besides forcing essentially an infinite categorical scheme onto the description forming process the existing schemes must make arbitrary distinctions between the description of a real world situation and the description of the terminology used to make a discription. We propose a simpler notion of description as a binding of facts and propose a language for constructing those descriptions.

In this section we discuss the importance of having a formal ontology as a basis for the design of a representation system which can support the acquisition, storage, and reasoning capabilities of KAMSS. We discuss the relevant work in the area of formal semantic theories and argue for the position that any system of the scope and flexibility of KAMSS must embody a methodology for generation (or at least selection) of a relevant semantic theory based on the task requirements. However such a capability while conceivable, is years away from implementation. We therefore present a theory of natural language semantics which can form a solid basis as a default theory on which to build a representation for a near term KAMSS. This theory is referred to as situation semantics. After a discussion of the basics of this theory we identify a number of limitations with the current state of this theory and suggest ways in which these limitations may be overcome.

We will focus the discussion of the basic ontology issues of KAMSS around the problem of understanding of utterances presented to KAMSS. The rationale behind this focus is that the capabilities required to understand and store the information in these utterances exceeds that of the other performance aspects of the KAMSS system. The types of utterances implied by the usage scenarios described in Section 3 are of an extent which far exceeds those normally accounted for in the traditional theories of semantics. For the purpose of the majority of this discussion, the term "utterance" will be used to refer to a set of symbols which communicate a usable quantum of information. Thus a packet may be a sentence, an embedded phrase, clause, or sentence within a sentence, a symbol on a sketch, or an entire passage of text. Discounting a sketch input, the natural language structures range over:

1. Indicative statements, including those with definite descriptions, and complex forms particularly conditionals. We will focus on the **use** of indicative statements as a mechanism to convey facts which describe a situation.

2. Questions, focusing on the types which query for a fact and those which postulate a situation.

3. Commands, focusing on those whose use is to bring about a situation.

The goal of this section is to present a unified framework for describing the interpretation of such packets within a known structure of referents as well as a discovery mechanism for acquiring interpretations in situations where the acquisition of the referents is the primary activity (actually the predominant phenomenon).

The underlying assumption of this section is that there is not one correct theory of meaning or interpretation. Rather such theories are to be considered as systems. Such systems may be well formed or ad hoc. They may be useful or academic. But for an intelligent agent to understand, communicate, and reason about his environment, he must have the capability to formulate/learn/adopt (or adapt) different theories at different times. We believe this capability to be primary to language skill acquisition. However, discussion of this last point is outside the scope of the current section.

The first step in the process of modeling simulation and decision making outlined in Section 2 is that of the customer understanding his (or her) manufacturing situation. There are several aspects of this "understanding" or knowledge about the manufacturing situation which are of interest. The first is the question of how the customer acquires or updates understanding. Another is related to the first, that being how a customer internalizes this understanding. The third aspect is how this understanding is conveyed, using language, to the analyst.

We will focus in this section on the issues of semantics relevant to the conveyance of knowledge about a manufacturing situation between a customer and an analyst. We will refer to the totality of such information as the "System Description (SD)." In this context, semantics refers to the delineation of what the customer "knows in knowing what utterances of his language mean" [Barwise and Perry 1983]. Implicit in this discussion we assume a type of system description which is radically different from the basic ideas behind formal system specifications in that first of all the system description is not assumed to be complete (i.e., SD is factual not actual) and that only a small portion of the sentences within these description are not *efficient.*[1]

It should be recognized that a system description as a representation of the customer's understanding of his manufacturing situation is necessarily incomplete. We have all experienced knowledge about a situation for which even natural language forms too narrow a bandwidth to communicate. Even when we extend our system descriptive language to include engineering drawings, renderings, and

[1] That is, most of the "claims about the world" which are made in a system description are dependent upon who made them and when they were made.

sketches, there are still important language mechanisms which the human to human communication system uses which are unavailable to use (such as gestures, inflection, tone, etc).

From the onset, we must recognize that the communication mechanisms upon which we will rely are faulty. We must also recognize that the retrieval mechanism of the customer and his language competency (i.e., the ability to translate what he knows into written text) have both individual and inherent limitations. Thus any representation which is directly constructed from a natural language formulated system description must be assumed to be factual but not actual. Any system which would attempt to capture such a description and use it as a basis for the design of a model to answer questions about the system must be capable of dealing with the inherent incompleteness of the description and must have the capability to both request additional information as well as to design around situations in which the missing data was unavailable (i.e., the customer does not know and cannot cannot easily find out). It is for this reason that we have attempted a more basic epistomology than is normally the case. We want the KAMSS to be able to use utterances as a source of perceptions, the classification of which (into the ontology) may be held up until further information is acquired. Any attempt to characterize the information content of a text requires the establishment of a world view or structure of reality. The approach to be presented takes a view which is strongly influenced by the works of [Ramey 1983] and [Barwise and Perry 1983] and previous work by this author [Mayer 1983].

5.1 Relevance of Consideration of Semantics

An interesting property of theories of semantics is that they are themselves systems (by the definition ascribed in Section 2). Semantics as a discipline is an area of study which takes as its goal the construction of theories of meaning and meaning related phenomena [Fodor 1977]. What will be the concern of this section is both "a" semantics (i.e. Situation Semantics) and the hypothesis that one must be concerned about a methodology for semantics development. The importance of "a" *theory of meaning* (i.e. "a" *semantic theory*) is that it allows us to be precise in our use of terms such as "a system description". The importance of a *methodology for developing a semantics* is that it provides insight into how to construct a mechanism which mimics how an individual understands, models, and communicates knowledge, observations, and experiences. In addressing "semantics" it should be recognized that a very dangerous undertaking is at hand. For, on the one hand, the intuitive (and widely used) notion of the term "semantics" used in reference to the meaning of symbols in a symbol system is that meaning which a large majority of people would ascribe to those symbols. However, the formal definition of what it is to provide a theory of semantics (often referred to as model semantics) is rooted in the construction of abstract set-theoretic based models on which the primitives of the symbol system are mapped in some consistent fashion. The basic mapping and the properties of the underlying model are then used to ascribe specific interpretations of the rest of the symbol set and of well formed combinations of those symbols.

By analogy, we may consider the user of the KAMSS to be the primary sensory mechanism for the KAMSS organism. As such, it is imperative that we understand the processing which goes on between that mechanism's perception and the language with which he communicates. We must also recognize the viewpoint of the KAMSS as an "intelligent" entity. The KAMSS unlike its human counter part has only one access to information from the outside world. That access is through the symbols input from its terminal device. When the KAMSS makes observations, it is making observations on the symbol stream coming to it or which has come to it over the terminal. Thus, the theory of semantics with which we are concerned must be viewed from the back side of the screen. The key question then is "What is KAMSS 'seeing' and what sense can KAMSS make of these observations?"

The practical relevance of study of a methodology for constructing a semantic theory comes from the need to represent the diversity of information communicated to the KAMSS through the discourse (or series of discourses) between the users of the system and the system itself. The utterances and renderings (sketches) have as their primary purpose the communication of information about the system, its problems, the desires of the user and so forth. If we are to capture this information in some form then we must have a structure for expressing the meaning of such expressions as:

1. The robo-carrier moves autobodies from the hot dip area to the paint area.

2. The robo-carrier is a type of induction motor conveyor with programmable path control, collision sensors and a 12 foot bed.

3. EAMRs are created by engineering for long lead time items in order to allow material sufficient acquisition time.

4. When the part is loaded the table is indexed and the machining is initiated.

5. What is the expected thru-put of this system?

6. Can this system be set-up to run at 80 jobs per hour?

7. Prepare a simulation model.

A large part of the analysis process of the human analyst is focused to disambiguation of basic understanding of the spoken words of the customer with the customer's understanding of those words. Sentences like #1 above carry both semantic information (that being the information about the relation between two objects in the real world) as well as pragmatic information (that there is an object in the real world referred to by the label "robo-carrier"). Sentences like #6 above can be interpreted as postulating or asking for the postulation of "possible worlds" which have certain desirable properties. Sentence #2 above is an example of a "description" of an object which can be interpreted as a collection of facts or as an extension of the basic concept structure of the underlying semantics.

The point of all this is that it is unlikely that a single semantic theory (even situation semantics) will account for all of the meaning and interpretation representation necessary to support a KAMSS. Therefore, we expect that the semantic theory must be flexible enough to be able to assign multiple interpretations to a syntactically unambiguous utterance, or else the KAMSS must be designed with the idea of having multiple semantic theories which it applies as required by the situation at hand. This requirement unfortunately rules out much of the formal work in model-theoretical and truth-conditional semantics which we had hoped to build upon as all of these systems are designed under a monotheoretic assumption. Even if a unifying theory was built its application within a practical KAMSS would include a constant extension of its basic structure. Thus, just as the KAMSS must incorporate the knowledge of an expert modeler to design models to answer questions posed against a system description, so also must it contain the knowledge of methods for formulating or at least extending its underlying basic semantic theory (or theories as the case may be).

The final rationale for investigations into the methodology of semantic theory development for the task at hand is to understand the influence on these theories of the assumed reasoning mechanism associated with the theory. It is important to know a priori what constraints such reasoning mechanisms might have on the meaning representation schemes and vice versa if we are going to be able to encode in KAMSS the intelligence to be able to select between competing semantics for a particular utterance in a particular situation.

The issues delineated in this section identified several characteristics (needs) which a theory of semantics must possess in order to be useful for the KAMSS. One other issue has to do with the storage efficiency of the possible representations required to denote instances of the SD semantics in a practical setting. Another issue has to do with the ease of extension of the theory. Not only should new meanings be easy to represent, but also new types of meanings. This implies that the chosen theory should support a discovery mechanism and not just a classification mechanism. A final important issue is the relation between the methodology for semantic theory development and the reasoning, and information access activities which must go on in the various modes of operation of the KAMSS. These turn out to be closely related but very different problems. The design of a flexible and extensible representation for encoding characterizations, definitions, and assertions is a separate issue from the use of these encodings in a particular reasoning process. Both of these tasks are distinct from the issue of indexing of individual structures in memory (or on disk) and the definition of areas for caching collections of these structures for task execution efficiency.

5.2 Orientation Relative to Existing Theories

Until only recently the issues of semantics were largely relegated to the study of philosophy, mathematics, logic, and theology. This section will not attempt to provide a complete overview of the evolution of theories of meaning nor even a complete classification of such theories. Such a treatise would be a major development in and of itself. Rather, the treatment of several of the major works which have influenced the thinking reflected in this section will be reviewed.

Theories of meaning can be broken into major classes. Traditional semantic theories focus on the meaning of symbols in a formal symbol system. Traditional semantics tended to focus on the problem of entailments between declarative sentences. The logical systems with which we are most familiar (e.g., propositional and predicate logics) are based in the tradition of sentence entailment manipulation [Barwise 1985b; Fodor 1977]. The lineage of this approach to semantics includes Frege, Russell, and Tarski. The methodology established in model theoretic semantic development which resulted from this work has served as a foundation for work in understanding and formalization of the semantics for a wide variety of formal languages including programming languages [Stoy 1985].

It is only recently that linguists joined the act. With the emergence of generative grammars came a ground swell amongst linguists to carve out a rightful claim for theories of meaning as a legitimate part of the study of language. The successes in generative grammars laid the groundwork for the belief that the natural languages did have underlying formalisms which would permit rigorous semantical analysis despite the apparent vagueness, ambiguity, and inconsistency of the surface forms. Their claim certainly has merit in that traditional theories of meaning focused almost exclusively on declarative statements (e.g., Socrates is a man. All men are mortal. Therefore, Socrates is mortal). Normally the traditional theories do not consider the "meaning" of commands, questions, or really even assertions (e.g. From – I assert that Socrates is a man. – and – I assert that all men are mortal. – it does not follow that – I assert that Socrates is mortal.) In fact the traditional logics focus exclusively on material implication. Armed with the tools of generative grammars, linguists embarked on a series of developments in the field of semantics with the work of such notables as Fodor, Katz, Montague, Lakoff, and Chomsky following the generative grammar theme. Another school Austin, Searle, (and later Katz) followed the speech acts theme.

The theory of meaning which has been most influential to our efforts is the "Situation Semantics" [Barwise and Perry 1983]. We are attracted to this system because the two basic underlying assumptions (that meaning is relational, and that meaning is distinct from interpretation) concur with our own intuitions resulting from observations made on system descriptions as communicated verbally and through written texts. Thus, we see this particular theory as applicable in a wide range of KAMSS activities. Situation semantics represents an instance of a semantic theory which is useful as a starting point for the development of a representation system for KAMSS. It is neither right nor wrong, only well formed and useful in a particular task. We believe that for an effective KAMSS to be constructed we must ultimately understand how to embody KAMSS with the capability to formulate semantic theories of its own. What will be presented in the following sections is a summarization of the major points of that theory, a reasonable notational representation of the basic principles of that system, some minor extensions, and a discussion of the implications of such a theory for knowledge representation and reasoning within the KAMSS.

5.3 Systematic Relative Meaning (SRM): A Methodology for Semantic Theory Development

This section is concerned with the description of the process by which semantic theories are developed. As such we are concerned with the process used by the philosophers, mathematicians, linguists, or AI researchers to evolve a semantic theory. It is our belief that, through the study of this methodology we will acquire insight into:

1. How to embody KAMSS with at least a rudimentary capability to extend or revise its underlying semantic theory.

2. How the person in a manufacturing domain comes to build an understanding of that domain.

3. How the manufacturing systems analyst can formulate models for problem solving in an unfamiliar domain.

Most often a semantic theory development process starts with the observation of a shortcoming or failure in an existing theory. For example, the apparent failure of Frege's principle of "Compositionality" in the following example was one of the issues that situation semantics was attempting to overcome. Given the following two statements:

1. John made Mary bake a cake.

2. John made Hitler invade Poland.

and given that the first is true then we can determine (using the substitution of equals and the principle of compositionality) that John should be held responsible for the invasion of Poland.

Such problems can be as simple as a "class clash" in which an observation fails to fit within an existing class structure, or as insidious as a determined inconsistency or incompleteness in the underlying axiomatic basis of an existing semantic theory as indicated above. In general, what is developed is a set of "case" situations which the current theory cannot accomodate. Initially an attempt is made to adapt the existing semantic theory in some natural way (such as to bend the rules of the existing method or to ignore the problem phenomena). However, if the problem situation becomes one of the important focuses of the effort then a new concept or construct must be introduced. This process normally proceeds from an abstraction of the case situations compiled. However, it may be driven from artifacts of the existing semantics formalism (i.e. insights provided by the math or logic basis of the existing theory.

After compilation of sufficient case situations to form a motivational basis for a new concept (Introspection) there are actually four steps to the introduction of a new concept [Benzon 1987]. The first is the characterization of that concept (Creativity). The second is the refinement of the characterization into a definitional form (Rationalization). The third is the introduction of the concept into the existing structure in a consistent fashion (Formalization). The fourth is the construction of proofs of consistency and completeness of the existing system augmented with the new concept (Discovery).

Relative to the Introspection step there is the inherent question of whether a KAMSS can be endowed with the capability to notice the uniformities. The essential confusion is not centered on the mechanistic ability (though algorithmic considerations must ultimately be considered) rather the issue of the formulation of the motivational triggers that would result in the initiation of the noticing of (or search for) similarity is the central issue.

The Creativity step is of specific interest to our method because of the current lack of understanding relative to the process by which phenomenologically naive concepts get transformed into abstract concepts. In the human experience phenomenologically naive concepts of groups of similar objects existed long before the abstract concept of a *set*. Even this example brings to light another issue, that being the fact that phenomenologically naive concepts can be both concrete (e.g. shiny hard metal) and abstract (e.g. groups of similar objects). Precisely how much of this background information is necessary to embody the KAMSS with an effective methodology for semantic theory construction is not known at this time.

Rationalization can initially be considered as a process of *Divide and Conquer*. Through this process axioms for membership are formulated. These axioms can be viewed as establishing links between the concept to be defined (definiendum) and the concepts used to define it (definiens). From a methodology point of view we note that the creation of these rationalizations has as its completion criterion the adequacy required by the theory in which they are used. We note that in most formal theories the property of functionality is considered to be an acceptable completion criterion. It is probable that this criterion can be used as an initial criterion in KAMSS. We also note that the rationalization process is recursive, the definiens having their own rationalizations to a point where a primitive of the theory is reached. From observation of this process we can also note that there are situations in which the axioms for membership are both necessary and sufficient. In these cases the definiendum and the definiens can be considered to be at the same level of abstraction. In other cases, the axioms of membership are somewhat weaker only providing necessary or sufficient conditions. In these cases the definiendum is generally considered to be at a higher level of abstraction than the definiens. In the cases where the definiens are at a higher level of abstraction than the definiendum an error in method is noted.

The Formalization process can be viewed as one of establishing the *axioms of membership* for a new concept. These axioms often go hand in hand with those for the Rationalization step. They are also driven by the theorem proving activities of the Discovery step.

In the following subsections we present a structure for characterization of the elements of a semantic theory which can be used as the basis for application of a reasoning process based around the methodology sketched above. In Section 6 we will illustrate the use of this classification scheme.

5.3.1 Basic Building Blocks

What we are proposing is a system of abstract objects which can be used in a method for development/extension of a semantic theory. These objects can be used to classify the component of a semantic theory. The basic building blocks of the methodology are properties, bindings, and stakes. The unique characteristic of the method is that we do not postulate a priori a specific instantiation of what constitutes a property or binding. In fact, the method presumes that the discourse itself, or the thought process of the individual person, establishes the grounding of these concepts. Thus, what may appear as a property in one frame of reference could very well appear as a binding of a collection of properties in another frame. Similarly, the label used to denote a binding in one frame may well appear as a property denotation in another frame.

5.3.1.1 Properties

For purposes of the methodology *properties* characterize the atomic elements of perception at a spatiotemporal location and discourse situation. The basic notion of a property is something perceived by an organism. Properties are the basis for organism attunement to uniformities. These elements are not assumed to be uniform across individuals nor even necessarily uniform within an individual. The cognitive/physiological processes involved in the discernment of properties is not an issue which we attempt to deal with. Nor are we necessarily concerned with whether or not the objects in the real world "possess" properties. It is the individuals' perception of reality which is of importance (not reality itself) in the understanding and common sense reasoning processes described in Section 2.

We leave it to the tool (instrument) builders to augment our perception capabilities. [2] We only note that whatever that process is, it apparently has the charac-

[2] The related term "attribute" seems to carry with it the notion of acquisition via a tool. For example, the term "age" when applied to an object is a characteristic which cannot be "observed" in the same sense as "color". In fact with out an associated "frame of reference" and a tool to manipulate/compare measurements in that frame the attribute cannot be perceived.

teristic of being able to operate in a recursive fashion. For example, I may refer to a property of an apple of being red, or of having a color property whose value is red. I may also refer to a property of red of reflecting light within a specified range of the electromagnetic spectrum or of being the result of the combination of certain physical substances.

Normally, properties enter the semantic theory as predicates of some sort. If the predicate is functional then it is always advisable to use a function. Otherwise, or if there is doubt a simple predicate will suffice.

5.3.1.2 Bindings

Bindings are considered to be "links between." Examples are, a link between two properties, between a property and itself, between any collection of linked properties or between links themselves. A linkage between a collection of properties which persists within the frame of reference of the discourse situation can be used to supply the referent to a label which denotes an object in the domain of discourse. Thus, a cluster of properties which remain more or less stable over time is taken as our perception of an individual, object, system, etc. Rather than taking the classical view that an object (individual, etc.) is anything to which properties can be ascribed, we view as primary the recognition of the properties and view the binding (the time persistent association) of a collection of those properties as the recognition of the object. What are traditionally referred to as "relations" between objects then become bindings between bindings of properties. This allows us to refer to links between relations (links between links between property links). It should be noted that we are not trying to say that objects are bindings, or that relations are bindings. Objects, relations and many other things exist in the real world. Thus, we do not ascribe to the metaphysical theory that reality is only in the mind. However for a semantic theory, classes of things which can be defined as types of properties and bindings provide the handles for representing such real world objects. These abstract objects (not the real ones) are what ultimately do the work for us in building representations and using those representations to perform inference. What we are interested in is how information about reality can be so structured that we can imitate the semantic theory discovery and extension mechanisms which are minimally required for operation of the KAMSS.

From a construction point of view concepts of the binding type can enter the semantic theory as either multi-place predicates or some similar structural element (for example as a type). From a methodological point of view it is important to note that when a concept has internal structure or can occur in many different forms this normally implies a binding. As will be seen in Section 6 we believe that this is one of the pragmatic semantic clues that a human analyst uses to extend his ontology.

5.3.1.3 Stakes: A basis for reference and ordering

Before talking about higher level constructs we must first address the issue of how to treat uniformities. To be attuned to a uniformity implies the need for referent and ordering concepts. We refer to instances of the first concept as a *stake*. The notions of time and place (or location) provide examples of two phenomenologically naive stakes. For example, it would appear that any notion of binding (or higher level concepts such as sets, relations, events, etc) must incorporate some element of time (an apple is not red when it is green or when it is rotten). The notion of assignment is another instance of the stake concept, especially from the point of view of the process of reasoning outlined above. Another important stake is the "individual". In Situations and Attitudes we see the use of this stake as a primitive concept which results in the inability of that semantic theory to individuate the parts of an individual. Stakes are nothing more than ways of identifying, collecting and ordering properties and bindings. Thus the notion of a set as a collection of objects which can be referred to as a unit is a type of a stake.

We look to the work on formalization of set theory for insight into the methodology for stake formulation in a semantic theory. In these works we find the following general categories of axioms required to axiomatize sets:

1. Identification Axioms,

2. Construction Axioms,

3. Existence Axioms,

4. Structural Axioms.

Identification axioms characterize the properties of the individuals which can participate in a "stake" (e.g. set), if you will the primitives relative to a stake. Construction axioms characterize the way the collections and orderings can be built up. In ZF set theory this includes the axioms of "Pairing, Powerset, Replacement, Separation, and Choice". In KPU this would include Δ_0-Collection and Separation and Pairing. Existence axioms normally describe the default case of no elements and the infinite element cases. Finally, the structural axioms cover the issue of what constitutes the extension of a stake and restrictions on the membership in a stake. The methodology we are proposing would require the formulation of each of these axiom classes during the formulation of a stake concept. In practice this is normally accomplished via the definition of a new stake in terms of an existing one (the most popular choice being "sets"). As we shall see in the context of continuous concepts this is potentially a dangerous approach.

The intent of the above outlined structure is to serve as the basis of a methodology for developing theories of semantics or more practically for the design of a knowledge component of KAMSS which could extend the built in semantics of the description acquisition and reasoning components of that system. With such a capability we do not need to restrict ourselves to speak to "a" theory of semantics. Rather, we can refer to a system of semantic theories interacting (and possibly evolving) during a discourse situation. This view of meaning implies that the individual deals not with a single theory of semantics, but rather is involved with a constant process of semantic theory generation.

The implication of this being that if we hope to duplicate an individual's understanding process we must be able to construct a system which is not based on a single semantic theory but rather one which has the capability to generate a useful theory of semantics for a particular situation. What this leads to is the conclusion that "reasoning" cannot be separated from semantics. Within this framework, the notion of reasoning can be viewed as a discovery activity. Given a perception (which may be of a sign or a symbol) the reasoning process must assume that that perception stands for something, or it discards it. It may "stand for" itself, a property, a binding, or a complex assemblage of such things (e.g., a situation). One feasible approach which the discovery process might take is then one of assigning to the perception initially as a "binding". Once the perception is assigned to a binding then the discovery process can pursue the investigation of the implications of that binding. If the binding appears to be "inconsistent" with other assignments which have been made or which are made as time proceeds, then the assignment may be abandoned and an alternative assignment attempted.

Note that this process in the case of the human experience is continuous. As previously noted, one of the difficulties in achieving convincing machine intelligence is that the machine's "life span" is generally measured in seconds or minutes. (Actually our existing knowledge based systems compare reasonably well intellectually against other organisms of this average life span.) Note also that the notion of consistency (or inconsistency) is predicated on the matching mechanism employed by what we will refer to as the "logic" mechanism. That is, we distinguish between reasoning as a discovery process and a "logic" which is a mechanism which supports that process. Traditionally the term "logic" refers to a "proof procedure" (e.g. the axioms and rules of inference). Where difficulty has arisen is in the confusion of the proof procedure (or mechanism) with the reasoning process it is intended to model. We believe that it is as crucial to separate these two concepts as it is to separate the interpretation of a sentence from the meaning of a sentence [Barwise 1984]. The need to distinguish "reasoning" from "logic" has been recognized by other researchers most notably Harman [Harman 1986]. However, most other views on this subject attempt to eliminate one or the other (e.g., Harman discredits logic, Kolwalski discounts reasoning.) Our view is that both are essential for the operation of an "intelligent" mechanism.

5.4 Situation Based Semantics

At the top level, situation semantics as a theory of semantics attempts to account for the following phenomena [Barwise and Perry 1985]:

1. Entailments between sentences:
   That is, the ability of statements in the form of sentences or sentence fragments to provide material implication of new knowledge.

2. External significance of language:
   Refering to the relation between language and the information carried by expressions in that language. The important distinctions to recognize are that:

2.1. Information is prior to language: This implies that a theory of meaning must have a way to represent the "way the world is" in order to denote the information that an utterance is intended to convey.

2.2. Information is carried by language: All kinds of utterances convey information. The restraint that traditional logics place on the meaning of a statement (its truth value) unreasonably limits the information that can be conveyed by that statement. [3]

2.3. Information is not meaning: Situational semantics proposes a relational theory of meaning that sees meaning as systematic relations between types of situations. It is the "meaning" of utterances which allows them to carry information. The methodology of situational semantics is to analyze the information conveyed to investigate the meaning of sentences. The extensions in this section attempt to extend the application of this methodology to utterances beyond simple indicative sentences.

3. The productivity of language:
The ability to understand an essentially infinite number of expressions formed from a finite set of words. The important aspect of productivity which situation semantics accounts for is the apparent failure of Frege's Principle of Compositionality. The solution is essentially the result of distinguishing between the meaning of an expression and its interpretation in a particular utterance.

[3] This point is particularly relevant to the KAMSS problem. Since we want to be able to represent the information conveyed in an interactive discourse which includes utterances other than simple indicative sentences. Also the goal of the system description capture is the assimilation of a representation of "how the system is" not what is the "truth" of the utterances produced. We would also like to have a uniform representation of this kind of information so that we can consistently handle both the implication of "the robo-carrier is an induction motor conveyor" and "induction motor conveyors are inaccurate positioning devices" as well as the conditional constraint [Barwise 1985a] "If the buffer size of the hot dip tank is not increased, a production bottleneck will occur at the loading station."

4. Efficiency of language:
   The ability of a particular utterance to be used at different times and places by different people to carry different information. The central point of this phenomena is that meaning underdetermines interpretation (the property of having the same meaning but different interpretations). Barwise and Perry [Barwise and Perry 1983] distinguish between linguistic meaning which is the interpretation of an expression fixed by the language and the "context of use" which is the interpretation of an expression fixed by its use. Factors of the context (sources of efficiency) that are exploited to get from linguistic meaning to use meaning include:

   4.1. Exploitation of the discourse situation through indexicality, which is the term for the dependence of the interpretation of an utterance on facts about the discourse situation.

   4.2. Speaker connections and reference which includes:

      4.2.1. Uses of names,

      4.2.2. Deictic use of pronouns-meaning of words like "that" to refer to things the speaker is connected to.

   4.3. Referential uses of tense,

   4.4. Exploitation of resource situations by their being:

      4.4.1. Perceived by the speaker,

      4.4.2. The object of common knowledge,

      4.4.3. The way the world is,

      4.4.4. Built up by a previous discourse.

5. The perspectival relativity of language evident from the facts that:

   5.1. Different speakers are in different discourse situtations.

   5.2. Different people have different causal connections to the world.

5.3. Different people have different resource situations available to them.

6. The ambiguity of language:
   By recognizing that expressions are uniformities across certain kinds of situations it can be seen that ignoring contextual references, connectivities, and resource situations causes apparent ambiguity.

7. The mental significance of language:
   This phenomenon refers to the fact that utterances carry information about the state of mind of the speaker. That is we can (and do) learn about the speaker by both the utterances he uses and the information he communicates. The important contribution which situation semantics makes in this regard is that external significance can be shown to have priority over mental significance in that mental significance is adequately explained by external significance.

Situation based semantics (a la Barwise and Perry) focuses on indicative statements formed by declarative sentences only, since utterances such as questions, commands, requests, and promises do not describe situations. The situation semantics theory is built upon a set of primitives which include individuals, relations, and spatiotemporal locations. These primitives are postulated to be the uniformities across real situations that a human is attuned to. In this view, reality consists of situations which consist of individuals having properties and standing in relations at various spatiotemporal locations. Utterances are assigned *interpretations* in virtue of the meaning of the sentence used, where the meaning of a sentence is defined as a binary relation between the type of situation where the sentence can be used, and the "type" of situation described by the use of the sentence in such a context.

Of course saying such uniformities exist without a way of talking about them would be rather useless. Thus, set-theoretic constructs are introduced to classify real situations, abstract out the primitives, and provide the structures for representing persistence. For reference, the following summary of the basic concepts and notation of situation based semantics is provided. This notation is slightly different than that of [Barwise and Perry 1983] — hopefully it is somewhat clearer.

Note that a theory of semantics proposes an abstact model which can be used to classify reality. In what follows it should be noted that unless otherwise noted when we refer to situations, courses of events, or other situation semantic objects we are referring to the "abstract" situation semantic object. Thus for example we will define the abstract course of events as a set. This abstract concept is not perceived nor are we stating that real courses of events are sets. Situation semantics views real situations as being prior to the uniformities that allow us to perceive those situations. The uniformities are viewed as being prior to the abstract situations that are provided by the theory to classify descriptions of the real situations. The uniformities chosen by situation semantics (i.e. individuals, relations, and spatio-temporal locations) are recognized as being a subset of the uniformities perceived by a real person. They were chosen as a judicious selection of the most important uniformities for explaining the characteristics of natural language described above (in particular the flow of information.) The uniformities provide the link between the real and the abstract situations or courses of events. Therefore, in what follows, we will at times distinguish between real situations and the abstract situations that classify them. However we will not distinguish between real and abstract individuals, properties, and relations. The reason for this is that the theory takes the point of view that the real situations are basic and that these primitives arise as perceptable uniformities across real situations.

5.4.1 Set-Theoretic Notation Conventions

1. Calligraphic capitals $\mathcal{A} \ldots \mathcal{Z}$ are used to represent collections.

2. Italic capitals $A ... Z$ are used to represent sets.

3. $\langle a, b, c, \ldots \rangle$ is used to indicate a sequence.

4. $\{a, b, c, \ldots\}$ is used to indicate a set.

5.4.2 Symbols Used as Variables to Represent Primitives

1. a, b, c, ... represent individuals.

2. $\mathcal{A}$ represents the collection of individuals.

3. r, r', r'', ... represent relations.

4. r_0 represent 0-ary relations, or situational states.

5. r_1 represent 1-ary relations, or properties.

6. r_2 represents 2-ary, or binary relations.

7. r_n represent n-ary relations.

8. $\mathcal{R}_n$ represents the collection of n-ary relations.

9. $\mathcal{R}$ represents the collection of all relations (i.e. $\cup_n \mathcal{R}_n$).

10. l, l', l'', ... represent spatiotemporal locations (i.e. instances of parts of the space-time continuum.

11. $\mathcal{L}$ represents the collection of all space-time locations.

5.4.3 Special Space-Time Relations

1. $l \prec l'$ l means wholly temporally precedes l'.

2. $l \circ l'$ l temporally overlaps l'.

3. $l@l'$ l spacially overlaps l'.

4. $l \subseteq_t l'$ l is temporally included in l'.

5. $l \subseteq_s l'$ l is spacially included in l'.

6. $l \subseteq l'$ l is temporally and spacially included in l'.

5.4.4 Situation Types

1. $\mathcal{Y} = \langle r, x_1, \ldots, x_n \rangle$ is a constituent sequence where r is an n-ary relation, and $x_1, \ldots, x_n$ are objects. [4]

[4] The term "object" is introduced as a meta-language construct to allow us to refer to any primitive as well as any complex situation semantic construct (e.g. a situation, a course of events etc.).

2. σ represents an extensional relation between constituent sequences and 0, 1 and "undefined" (i.e. a set of pairs $\langle \mathcal{Y}, 0 \mid 1 \rangle$). Such a relation is called a *situation-type*. The relation $\sigma : \{\langle\langle r, a, b\rangle, 1\rangle, \langle\langle r', a, c, \rangle, 0\rangle\}$ is indicated by:

$$\begin{aligned} \sigma :=& r, a, b;\ \text{yes} \\ & r', a, c;\ \text{no} \\ & \vdots \end{aligned}$$

The relation $\sigma : \{\langle\langle r, a, b\rangle, 1\rangle, \langle\langle r', a, c, \rangle, 0\rangle\}$ where neither $\langle\langle r'', a, d, \rangle, 0\rangle$ nor $\langle\langle r'', a, d, \rangle, 1\rangle$ is indicated by:

$$\begin{aligned} \sigma :=& r, a, b;\ \text{yes} \\ & r', a, c;\ \text{no} \\ & r'', a, d;\ \text{undefined} \end{aligned}$$

3. The "in" notation is used to indicate that a set of pairs is a subset of σ:

$$\begin{aligned} \text{in } \sigma :& r, a, b;\ \text{yes} \\ & r', a, c;\ \text{no} \end{aligned}$$

4. A situation type σ is coherent if:

   4.1. in $\sigma : r, x_1, \ldots, x_n$; yes then not in $\sigma : r, x_1, \ldots, x_n$; no

   4.2. in σ : $same, a, b$; yes then $a = b$ (i.e. σ does not represent two different objects as being the same)

   4.3. not in σ : $same, a, a$; no (i.e. σ does not represent anything as being different from itself)

5. Two situation types are *compatible* if their union is coherent. $\sigma \psi \sigma'$ is the notation for *compatible*(σ, σ') which is true iff *coherent*$(\sigma \cup \sigma')$.

5.4.5 States of Affairs

A state of affairs s is a pair $\langle l, \sigma \rangle$. States of affairs have two important properties (i.e. *factual* and *actual*) and can stand in two basic relations with other states of affairs (i.e. *part-of* and *contained-in*). Informally, a state of affairs is *factual* if all of the facts in it are correct as far as it goes. On the other hand for a state of affairs to be actual it must be exhaustive. As can be seen by the following definitions of *factual* and *actual* properties a situation type of a *factual* or *actual* state of affairs is coherent.

1. $factual(s)$ if:
   if in $s:\ \ r, a_1, \ldots, a_n;$ yes
   then at l,$a_1, \ldots, a_n$ stand in the relation r.
   and
   if in $s :r, a_1, \ldots, a_n;$ no
   then at l,$a_1, \ldots, a_n$ fail to stand in the relation r.

2. $actual(s)$ if:
   if at l, $a_1, \ldots, a_n$ stand in the relation r.
   then in $s :r, a_1, \ldots, a_n;$ yes
   and
   if at l,$a_1, \ldots, a_n$ fail to stand in the relation r
   then in $s :r, a_1, \ldots, a_n;$ no.

3. $part-of(s, s')$ if $l_s = l'_s$ and $\sigma_s \subseteq \sigma'_s$.

4. $contained-in(s, s')$ if $\sigma_s \subseteq \sigma'_s$.

5.4.6 Courses of Events

1. The set $e = \{\langle l_1, \mathcal{Y}_1, i_1 \rangle, \ldots, \langle l_n, \mathcal{Y}_n, i_n \rangle\}$ where any i_k can assume a value of 0, 1 or undefined, is called a course of events (coe).

2. $\mathcal{E}$ denotes the collection of all coe's.

3. e^* is a function defined by $l \in$ domain (e^*) iff $\{l, \mathcal{Y}, i\} \in e$ for some $\mathcal{Y}, i$. Thus e^* is a function from locations to situation types.

4. For $l \in$ domain (e^*) $e^*(l) = \{\langle \mathcal{Y}, i\rangle : \langle l, \mathcal{Y}, i\rangle \in e\}$. Thus e^* can be thought of as a set of s's (states of affairs).

5. e_0 ia a part-of e_1, $e_0 \subseteq e_1$ iff $\forall s \in e_0^*$, $s \in e_1^*$.

6. Compatibility between courses of events is defined as follows:
$e\psi e'$ iff $\forall l$ $\in$ domain e and e' then if σ is assigned to l by e^* and if σ' is assigned to l by e'^* then $\sigma\psi\sigma'$. e and e' are said to be *compatible.*

7. A course of events is *coherent* if it only assigns coherent situation types to the locations in its domain. Thus a *coherent* course of events is *compatible* with itself.

5.4.7 Structures of Situations

It will become necessary in the presentation which follows to have a characterization of *factual* and *actual* which is independent of reference to real world sitations, uniformities or courses of events. The concept of a structure of situations as follows provides such a characterization. A structure of situations M consists of two elements each being a collection of *coe's*.[5] $M{=}(\mathcal{M}, \mathcal{M}_0)$ where $\mathcal{M}_0$ is a subcollection of $\mathcal{M}$ with the following properties:

1. Every $e \in \mathcal{M}_0$ is coherent.

2. If $e \in \mathcal{M}_0$ and $e_0 \subseteq e$ then $e_0 \in \mathcal{M}$.

3. If X is a sub*set* of $\mathcal{M}$ then $\exists$ a *coe* e in $\mathcal{M}_0$ such that for every *coe* e_0 in X $e_0 \subseteq e$.

4. If C is a *constraint* in $\mathcal{M}$ then $\mathcal{M}$ *respects* C. [6]

---

[5] Actually it may be clearer to think of the structure as being a collection of *coe's* denoted by $\mathcal{M}$ which has a distinguished non-empty subcollection $\mathcal{M}_I$. Either way the general idea is that we can construct a purely abstract structure which mimics the way *factual* and *actual* properties work in the real world.

[6] The terminology of *constraint* and *respect* used in this definition will be explained in Section 5.4.12 after we have developed some other required

Using a structure of situations as a context, the coe's in $\mathcal{M}_0$ are said to be *actual* and those in $\mathcal{M}$ are said to be *factual.*

5.4.8 Classifiers and Worlds

Abstract courses of events can be used to classify real world events if the following *classifies* relationship holds:

Let $\mathcal{E}_0$ be a collection of real world events. If $\mathbf{e} \in \mathcal{E}_0$ then e *classifies* **e** iff

if, in e, at $l, r, a_1, \ldots, a_n$; yes
 then, in **e**, at $l, r, a_1, \ldots, a_n$ stand in relation r;
 and
if, in e, at $l, r, a_1, \ldots, a_n$; no
 then, in **e**, at $l, r, a_1, \ldots, a_n$ do not stand in relation r;

Using the *classifies* relation we can define two important characteristics of abstract courses of events as follows:

1. e is *factual* if $classifies(e,\mathbf{e})$ for some real world event **e**.
2. e is *actual* if $classifies(e,\mathbf{e})$ and if $\forall e' \mid classifies(e',\mathbf{e})$ then $e' \subseteq e$. [7]

A *coe* w in a structure of situations M is a world of M if $\forall e$ in M $e \subseteq w$.

5.4.9 Persistence of Information and Informational Relations

The concepts introduced to this point merely allow us to classify real world situations and courses of events. In order to talk about information transfer between two situated agents we will have to extend the notion of coe's to develop a more powerful concept of an event type. The event type will allow us to describe how a situation holds information. But before we introduce that concept we will first

concepts. They are used here to allow a complete specification of a structure of situations.

[7] The symbol "|" is used as shorthand for "such that".

characterize the basic informational relationship between two coe's. In order to do that the notion of persistence is defined in the following manner.

Let $\mathcal{P}$ be a collection of *coe*'s; then $\mathcal{P}$ is (*simply*) *persistent* if $\forall e, e'$ if $e \in \mathcal{P}$ and $e \subseteq e'$, then $e' \in \mathcal{P}$. Persistent collections of *coe*'s form a topology (i.e. a family closed under unions, finite intersections, containing the empty collection and the collection of all *coe*'s) called the *topology of partial information.*[8]

Let m be a binary relation on a collection of *coe*'s $\mathcal{F}$. Let M be a situation structure with $\mathcal{M}$ as above. Since $\mathcal{F}$ is a collection m is a collection of ordered pairs of *coe*'s. m is *informational* on M if $\forall e \mid factual(e) \wedge e \in \mathcal{M} \cap \mathcal{F}$ then $\exists e' \in \mathcal{M} \mid m(e, e')$. For each $e \in \mathcal{F}$ let $[\![e]\!]_m = \{e' : m(e, e')\}$. If m is informational, then if e is factual then there is at least one $e' \in [\![e]\!]_m$, such that e' is factual.

Thus m is *informational* on M if $e \in \mathcal{M} \cap \mathcal{F} \Rightarrow [\![e]\!]_m \cap \mathcal{M} \neq \emptyset$. If $[\![e]\!]_m$ is *persistent* then the *informational* relation is said to be *persistent.*

A short example from [Barwise and Perry 1983] is appropriate here to illustrate how these concepts are intended to work. "Consider an event u, an utterance in which Sarah says to Jonny, "A dog is biting Molly," conveying information about an actual event e', one where Jackie is biting Molly at a present location l. Here take $\mathcal{F}$ to be the collection of assertive utterances and the relation m as the relation between an accurate assertion x and the situations x' it describes. Then $[\![u]\!]_m$ consists of all those e'' such that for some a:

$$\text{in } e'' : biting, a, Molly; \text{ yes}$$

$$dog, a; \text{ yes}$$

[8] Note that $\mathcal{P}$ can (and for most interesting $\mathcal{P}'s$ will) contain incompatible courses of events. Note also that such collections can get larger and more detailed in the sense of how many locations and detail at each location one wishes to capture. If this seems "uncontrolled" at this point, it is. However, it is all we need to get to the notion of informational relations between coe's and the ambiguity will be cleared up later with the notion of roles.

The actual event e' is one of these, and $[\![u]\!]_m$ is simply persistent.[9]

5.4.10 Indeterminates, Event Types, and Roles

Indeterminates are introduced to allow us to construct abstract event types and roles. The latter constructs are required in order for us to account for relations between situations that hold at different spaciotemporal locations or that involve different relations or individuals. There are three types of *basic indeterminates.* Indeterminates provide us with a variable like concept that allows us to make general statements about a "type-of" situation.

$\dot{\mathbf{a}}$, $\dot{\mathbf{b}}$,...indicates individual indeterminates.

$\dot{\mathbf{r}}$, $\dot{\mathbf{s}}$,...indicates relation indeterminates.

$\dot{\mathbf{l}}$, $\dot{\mathbf{l}}$',...indicates spatiotemporal indeterminates.

An Event Type is a *coe* where individual, relation, and spatiotemporal indeterminates may be present. An event type is denoted by $E, E', \ldots$. If $\dot{\mathbf{a}}$,$\dot{\mathbf{r}}$,$\dot{\mathbf{l}}$ are indeterminates in E this is denoted as $E(\dot{\mathbf{a}}, \dot{\mathbf{r}}, \dot{\mathbf{l}})$.

An Event Type with exactly one individual indeterminate is called an object type and is denoted by $O(\dot{\mathbf{a}})$. An Event Type in exactly one individual indeterminate and one location indeterminate $E(\dot{\mathbf{a}}, \dot{\mathbf{l}})$ is called a *complex property.*

A function which assigns individuals, relations, or spatiotemporal locations to indeterminates in an E is called an *anchor* for E. From an event type E and an anchor f we can construct another event type where each indeterminate in the domain of f is replaced in E by its value $f(\dot{\mathbf{a}})$. This event type is denoted by $E[f]$. If f is defined on all the indeterminates in E, it is called a *total anchor* for E. Note that if f is a *total anchor* of E then $E[f]$ is a course of events. A *coe* e is *of-type* E if $\exists f \mid E[f]$ *part-of* e.

[9] Since the definition of the "informational" property of a relation is dependent on the choice of $\mathcal{F}$ any change in $\mathcal{F}$ may cause the relation to be either not informational or not persistent. This might lead one to strengthen the notion of persistence to include a domain for the e' in the definition of persistence.

We would like to also classify relations between individuals (or locations or relations) and event types. For example to conveniently represent the relation between the buyer and seller in a purchasing event. One way to do this would be to define equivalence classes on collections of event types. Another cleaner approach is to extend the concept of *indeterminate* to define event types in terms of indeterminates and to use event types in the definition of indeterminates. Barwise and Perry chose the latter approach.

Every basic indeterminate is an indeterminate. If $\dot{\mathrm{x}}$ is an indeterminate and $E(\dot{\mathrm{x}}...)$ then the ordered pair $\langle\dot{\mathrm{x}},E\rangle$ is an indeterminate called a role (it is often also referred to as a *complex indeterminate*). There appears to be no consistent denotation for a role, however $\dot{\mathrm{x}}_E$ is often used. I prefer to use $\dot{\rho}$, when the event type and indeterminate are obvious from the context of use.

Given the above extended definition of the notion of an event type and indeterminate we need to revamp slightly the definition of what it means for a partial function to serve as an anchor for a role. We want to make sure that since an event type may be defined in terms of roles that the anchoring of the indeterminates is consistent. Therefore, a partial function f from either basic or complex indeterminates to individuals, locations, and relations is an **anchor** provided that:

1. For every basic individual, location, or relation indeterminate $\dot{\mathrm{x}}$ in the domain of f, $f(\dot{\mathrm{x}})$ is an individual, location, or relation respectively.

2. For every role $\dot{\rho} = \langle\dot{\mathrm{x}},E\rangle$ in the domain of f, f is an anchor for each indeterminate in E and $f(\dot{\rho}) = f(\dot{\mathrm{x}})$.

Note that an anchor f for a role $\dot{\rho} = \langle\dot{\mathrm{x}},E\rangle$ is said to anchor $\dot{\rho}$ in e if $E[f]$ is part of e.

Given a role $\dot{\rho} = \langle\dot{\mathrm{x}},E\rangle$, an object x *has-role* $\dot{\rho}$ in a coe e of type E iff

1. e is *of-type* E

2. for every total anchor f for E such that $E[f]$ is *part-of* e then $f(\dot{\mathrm{x}}) = x$.

This is denoted $\dot{\rho}_e = x$. Essentially it means that the only ways in which e is of type E have $\dot{\mathbf{x}}$ anchored to x.

Given an event type $E(\dot{\rho}_1, \ldots, \dot{\rho}_n, \ldots)$ and a coe e *of-type* E and such that every role $\dot{\rho}_i$ is uniquely defined in e,[10] then e is a *context-for* E with respect to $\dot{\rho}_1, \ldots, \dot{\rho}_n$. Contexts are used to describe uniformities across roles (e.g. as in the case of the type of event where the victim sues the hitter at some time later than the time of the hitting.)

5.4.11 Indexed Event Types

Indexed Event Types allow the representation of perspectives. For example, the difference between the perspective of Anna and Simon in virtue of the different roles that they play in the event

in e : at l, hitting, Anna, Simon; yes.

They also allow us to classify *abilities* of organisms, such as the ability of an individual to tell what position he is in at any given time. Such a classification scheme is provided by the systematic use of event types with distinguished indeterminates. We specify certain roles as fixing the meaning of certain indeterminates. Once these are fixed, any other event types in these indeterminates are called *indexed event types*. Thus, any indexed event type determines a family of event types indexed by the *contexts* for that event type. An indexed event type is a set of event types indexed by contexts $\{E_e\colon e \text{ is a } \textit{context-for } E\}$ and defined on a set of roles where all of the constituent roles are uniquely filled. Thus, for example, a condition like:

E: at $\dot{\mathbf{h}}$, sitting, $\dot{\mathbf{i}}$; yes.

[10] By uniquely defined we mean that there is an anchor f for E in e such that for any other anchor g for E in e, $g(\dot{\rho}_i) = f(\dot{\rho}_i)$ for all i.

relative to a context (e.g. a coe e) which supplies an agent and a location characterizes the ability of that agent to tell what position he is in.

We denote an instance of an indexed event type by $E_m[\dot{\rho}_1, \ldots, \dot{\rho}_n, \ldots]$ where $\dot{\rho}_i$ are the indexing roles, and m is a context which supplies unique values for those roles.

5.4.12 Schemata

A schema is a finite set of event types $S = \{E_1, \ldots, E_n\}$. An anchor for a schema S is an anchor for the event types in S such that if f is an anchor for S then $S[f]$ is the set of all event types $E[f]$ for $E \in S$. An anchor is a total anchor of S if it is a total anchor for each $E \in S$. A coe e is of type S if it is of type $E[f]$ for some $E \in S$ and for some f.

An indexed schema is a set of schemata defined on roles.

Schemata are introduced into the formal concepts of situation semantics for a variety of purposes, one of which is to handle the characterization of events that are in conflict. In order to capture the notion of conflicting event types we first introduce the notion of *negation* of an event type and the definition of negation of a schema. Given a simple event type E (i.e. one with no proper parts (no roles please)) we use $\neg E$ to denote the result of replacing 1 by 0 and 0 by 1 in the constituents of that event type. If E is not simple, we define $\neg E$ to be the schema:

$$\{\neg E_0 | E_0 \text{ is a simple part of } E\}.$$

Given a schema S we define $\neg S$ to be the event type which results from the union of the negation of each of the event types in S.

5.4.13 Necessary, Nomic, Conventional, and Conditional Constraints

Constraints are introduced to allow the representation of the fact that one situation may contain information about another situation. It is the attunement to these constraints which allows an individual to acquire information from one situation about another situation.

A distinguished relation *involves* is introduced in order to serve as the basis for constraint specification between event types. Informally, a type of event E *involves* E' if every actual coe e of type E is part of an actual coe e' of type E'. In general a simple constraint is a state of affairs of the form:

at l : involves, E, S; yes

where E is an event type and S is a schema.

Thus a constraint is a course of events with only event types and schemata and the relation of *involves* in its domain.

Let r be the *involves* relation. Given a simple constraint $C = \{\langle l, \langle r, E, S\rangle, 1\rangle\}$:

1. A coe e_0 is *meaningful* wrt C if e_0 is *of-type* E.

2. If a coe e_0 is *meaningful* wrt C then e_1 is a *meaningful option from* e_0 with respect to C if for every total anchor f of E, if e_0 is *of-type* $E[f]$ then e_1 is *of-type* $S[f]$. This is denoted by $e_0 {r_{mo}}^C e_1$.

3. If a coe e_0 is *meaningful* wrt C then e_0 precludes an event e_1 with respect to C if:

   3.1. $\neg e_0 \psi e_1$ or

   3.2. $\exists f \mid e_0$ is *of-type* $E[f]$ and for every extension of f to an anchor g for the indeterminates in S, e_1 is *of-type* $\neg S[g]$.

Given a simple constraint C a structure of situations M *respects* C if for every actual $e_0 \in$M, if $e_0 \in \mathcal{F}_C$ then $\exists e_1 \mid e_0 {r_{mo}}^C e_1$ and e_1 is *factual* in M. This is denoted M_C.

Given an arbitrary constraint C, M_C if M respects each simple part of C.

Necessary Constraints are those which arise from necessary relations among the properties and relations that we are attuned to.

Nomic Constraints are those which arise from natural phenomena, (e.g. laws of nature).

Conventional Constraints are those that arise from explicit or implicit conventions that hold within a community of living organisms.

Conditional Constraints are necessary, nomic, or conventional constraints which are known to hold under certain conditions.

5.4.14 Constraint Types and Indexed Constraint Types

The final major concept to be presented is the mechanism provided by situation semantics for classification of uniformities across constraints. Constraint types allow us to represent the *type* of constraint one gets by anchoring one of the indeterminates to some individual, relation, or spatiotemporal location. A contraint in which the event type and schema are anchored to an indeterminate is called a constraint type.

A constraint type is denoted by $C/\dot{\mathrm{x}}$ where $\dot{\mathrm{x}}$ is an indeterminate or set of indeterminates. In specifying a constraint type $C/\dot{\mathrm{x}}$ the following notation is used to indicate that the indeterminate $\dot{\mathrm{x}}$ must be anchored to some individual, location, or relation before we get the event type E and the schema S such that E involves S:

$$C/\dot{\mathrm{x}} : \text{at } l : \text{involves}/\dot{\mathrm{x}};\ E,\ S;\ \text{yes}.$$

If the indeterminate (or set of indeterminates) that participate in the constraint type definition have been assigned a special role across the participating event types then the constraint type is called an *indexed constraint type* $C/\dot{p}$.

The following subsections of this section build the KAMSS ontology by application and extension of the above presented situation theory. We honor the underlying metatheory assumptions of individuals, relations, and spatiotemporal locatives as uniformities across real situations. However, we recommend a revision of the set of abstract primitives to support the process of acquisition and manipulation of those uniformities. Within this context we argue for the adaptation of ensemble theory as the framework for providing set definition/manipulation capabilities for finite collections of individual objects, and a compatible set of capabilities for continuous concepts/substances. After that we present some of the basic ontology which is required for KAMSS acquisition, reasoning and modeling functions. Finally we provide an illustration of the type of acquisition capabilities enabled by our approach.

5.5 Problems with Existing Theories

One of the basic problems with many of the traditional semantic theories of is that they tend to follow the formal logic paradigm of making the interpretation of the primitives irrelevant to the argument at hand. Sentences in first order logic follow the Frege tradition [Frege 1960] in that the reference of a sentence is its truth value. But we want sentences to denote other things besides truth values (e.g., themselves, situations, relations between situations, bindings, properties, or any higher level assemblage of these notions). This is bothersome for two reasons. First, from the point of view of the fact that the use of argument in the KAMSS is an attempt to increase the knowledge base of known facts. Secondly, because the essential feature of the way humans use language is to refer to objects and situations beyond themselves. Of course humans have a distinct advantage over a computerized KAMSS. They have the ability and physical equipment to directly experience many of the phenomena which they express through their language. The KAMSS can only rely on what it has been told. Also the human experience base extends over a wide range of space time intervals. Situational Semantics as outlined above overcomes this problem by making the distinction between the interpretation of a statement and the evaluation of its truth value. In order to account for the characteristics of languages described in the last section situational semantics takes as the interpretation of an utterance the collection of situations referred to by that utterance. We will adhere to this principle in our SRM described in the following sections.

The theory of situations and attitudes as described in [Barwise 1984] can be used as the basis for our work. However, this theory is incomplete in many ways for the needs of KAMSS. Extensions required to overcome these deficiencies include:

1. Decomposition of the *individual* primitive to allow the representation of the process of attunement to uniformities of properties and property values across spatiotemporal situations as the basis for the recognition of entities.

2. Allowing individualization of parts of individuals (i.e. John's arm is a part of John). This includes accounting for continuous and quasi-continuous individuals.

3. Extension of conditional constraints to account for both *if–then* and *when–then* structures.

4. Accounting for generalizations and specializations.

5. Adding to the theory of attunement a consistent account of action.

6. Enhancement of the notion of attitudes to account for three levels of knowledge possession.

7. Provide a basis for reliable acquisition of knowledge as a precondition to behavior enabling possession of knowledge.

8. Provide a more general handling of the concepts of uniformity referents, (referred to as "stakes" in the following sections). Initially this includes the splitting out of space and time into separately analyzed phenomena.

9. Handling of location referents.

10. Handling of plural pronouns.

11. Allowing for sentence negation.

12. Handling of counterfactuals.

13. The use of attitudes to describe what a customer believes or thinks about his system.

14. How the process of utterance construction can be used to resolve ambiguity or acquire meaning.

15. Differentiation of the information displayed in an utterance versus the information conveyed by that utterance.

16. Adaptation of the discourse situation to characterize the man machine interface (i.e. where the theory of the listener and the speaker must account for the fact that these roles are alternatively played by the inanimate computer).

In the following sections we provide an epistemology which can be used as the basis for a meta-semantics (SRM). We then show the use of this structure in the adaptation of situation semantics to cover several of the issues identified above. Specifically we will focus on the accounting for actions through and extension of the event concept to a form which is capable of handling the meaning of utterances provided to or generated by the KAMSS environment described in Section 3. The development of the set-theoretic models necessary to extend the actual situation semantics will not be developed in detail. Rather we will point out where relevant extensions need to be made and what form these extensions are likely to take.

5.6 The Base Ontology

Given the above epistemology the following sections provide a reformulation / extension of some of the more important basic ontological components required for representing manufacturing system descriptions in KAMSS. We refer to this portion of the ontology as basic to distinguish it from the more domain specific concepts such as parts, machines, organizations, etc. It is useful to consider the analogy that can be made between the notion of an ontology and that of a theory. A theory is normally built upon a set of primitives with determinable properties, operations on those primitives and a set of axioms. An ontology on the other hand is built on a collection of **characterized** primitives (i.e. abstractions with nondeterminable properties), relations which can hold between such primitives, and implications which can be drawn from the relations. Thus use of theories is quite often directed at the formulations of theorems which can be derived from the axiom set. The use of ontologies is more often for the classification of observations in the pursuit of a theory. Hence the focus of the ontology (as pointed out in Section 4) is on the discovery of primitives and relational implications than on the proof of theorems. In this section we will characterize some of the basic intuitions important to an ontology to support the KAMSS concept. In the final subsection of this section will discuss the language issues associated with implementation of not only the following ontology fragment but also of the domain specific ontology and those relevant to the application of these ontologies in the KAMSS system.

5.6.1 Ensembles

Throughout the above description of the primitives no mention was made of operations other than those of perception. To get from the presented basis to traditional semantics based on set-theoretic concepts one needs an axiomatization of set theory. This is normally included in the epistemology itself. We have deferred its presentation to the ontology to allow for multiple definitions to be used. The motivation behind this approach is not that we are particularly interested in evaluation of alternative model theoretic semantics but that:

1. We want to allow for other definitions of a set, particularly we want to be able to introduce ensemble theory [Bunt 1985] for dealing with continuous concepts.

2. We want to explore ontologies built upon characterized structures rather than definitional structures.

Relative to the second point we add to our ontology the notion of an ensemble taken directly from [Bunt 1985] as a mechanism for handling continuous and "quasi-continuous" concepts. Such concepts play an important role in our every day life and certainly in understandings and descriptions of manufacturing systems. We find references to "flow shops", parts "flowing through" an area, spatiotemporal references such as an operation being "up stream" or "down stream" from a reference point. Also many of the processes which do occur are of a continuous nature (e.g. parts heating in a furnace, tools wearing, grinding operations and so forth.) Most importantly many of the cognitive activities in engineering and manufacturing have a continuous nature (e.g. designing, problem solving, modeling etc.). To have an ontology sufficiently robust to serve as the basis for representing these sorts of phenomena in a natural way means that we must deal directly with the continuous and quasi-continuous concepts.

Bunt in [Bunt 1985] reviews the various approaches put forth for dealing with continuous concepts in the context of traditional set theory. The basic problem with these approaches is that continuous concepts have a part-whole relation (they do have internal structure) but they do not have any atomic parts. Ensemble theory introduces the notion of *atomic ensembles.* A *unicle-whole* relation is introduced to describe the internal structure of *atomic ensembles.* Ensembles then can be classified as:

1. Atomic (i.e. having no parts),
2. Discrete (i.e. built up of atomic parts and indistinguishable from sets),
3. Continuous (i.e. not empty and each of its nonempty parts has a genuine part),
4. Mixed (i.e. having both a discrete and a continuous part).

Continuous ensembles have no atomic parts hence they have no members. Note that there is a difference between atomic ensembles and continuous ensembles An atomic ensemble has no proper parts whereas a continuous ensemble has proper parts (both of course are non-empty).

The traditional subset relation is subsumed as a special case of the part-whole relation applied to discrete ensembles. Similarly there is a merge relation which reduces to union and an overlap relation which reduces to intersection. There is only one empty ensemble which stands in the part-of relation to every ensemble. Finally, a notion of "completion" is introduced as the analog of set complement.

One of the advantages of the ensemble theory is that one can easily switch from a continous to a discrete view. In fact, Bunt illustrates that with the appropriate language design the switch consists of purely lexical substitutions. Thus ensembles provide us a way of representing such naturally continuous concepts as "the anodizer in the hot dip tank". It also allows the representation of "autobodies in the paint area" in either a continous or a discrete view (an important capability when addressing the task of model design).

Ensemble theory in the form proposed by Bunt does not solve all of the problems associated with continuous concepts. The focus of his efforts (the emperical observations behind his intuitions) is centered on concepts involving substances (e.g. liquids, materials, gases). There are other continuous concepts which do not behave appropriately according to the prescribed part-whole relation as defined by Bunt. Such concepts include:

1. part forms,

2. part features,

3. concept types.

A part feature (such as a character line in a hood or the belt line on an automobile) has continuity to it which does not satisfy either of the conditions for membership for atomic or continuous concepts. The reason for this is that three stakes if you will are used in the characterization of the essence of the feature (both the substance and the location and the relations of both to other parts of the substance). One of the interesting characteristics of these types of continuities is that they cannot be contained without containing parts of other features! Unlike the proverbial "cup of coffee" or "lump of gold" the "wind-split line" blends with the hood surface and may meet the "cowl edge" on one end and follow the closure surface on the other. However the approach he establishes with respect to handling substances can be used as a guide to establishing extensions which handle these issues.

5.6.2 Measures

Consider the information in: *For each additional part in the drying area add to the batch cycle extra time equal to its surface area multiplied by a factor of .5 plus the standard allocation of 3 minutes.* For an ontology to support the representation of quantified information over both continuous (e.g. time, space, etc.) and discrete (or mixed) concepts a notion of measuring must be introduced. There are a number of useful ways to establish such a measure system [Allen and Wright 1983; Hobbs et al. 1985; Hobbs et al. 1987; Bunt 1985; Corynen 1975] however we will follow Bunts' lead as what we are interested in is a way of *measuring amounts* of a continuum. The primary difference between *measuring* and *counting* (as it is used to determine the cardinality of a set) is in the introduction of the concept of a **unit**. Each unit concept has a category which would be a "stake" in the above described methodology. Units of the same category define a "system of measure" referred to as a **dimension**. Given a dimension, two of the units in that dimension and the numerical relation between them we can define an equivalence relation. A measure function is then easily constructed which respects the equivalence classes defined by that relation. The result of application of that measure function to an ensemble then is a value and a unit, an equivalence class that the ensemble is mapped into.

In general the approach we propose is one which uses the ensemble approach with a metric (as in Bunt) with one additional concept. The concept which is added to the system of measure defined previously as a *dimension* is that of a *resolution*. The combination of a dimension and a resolution constitutes a system of measurement. Thus, rather than having arbitrary transitions from discrete to continuous treatment of measurable quantities (such as space and time) as originally proposed by Bunt we have a determinable way to treat the transition from continuous to discrete representation. The resolution of a system of measurement determines a limiting value below which the value of the measure function is indeterminate. It is important to note that the value is not 0, it simply cannot be established. We define a "measurement" function which takes the value and units returned by a "measure" function and combines with that value the "deviation" which is equal to plus or minus the resolution for values greater than the resolution and only returns the resolution if the value is known to be different from 0 but less than the resolution.

5.6.3 Individualization of Parts of Individuals

Given the above described primitives of properties, bindings, and stakes and the capability to objectify over these concepts the notion of the individualization of parts of the individual becomes a natural consequence of the theory. An individual is in fact a binding of a set of properties which we can refer to with a label. The individualization of the parts of the individual is merely a binding over a subcollection of properties which is itself then bound to the original binding. This implies the fact that:

$$\begin{array}{l} \text{in } \sigma : r, a, b; \text{ yes} \\ \quad\; r', a, c; \text{ no} \end{array}$$

is not affected if *b* is itself a binding.

5.6.4 Space and Time

While it may seem strange to include in a discussion of the basic concepts behind an ontology for system description a discussion of "space and time", these two concepts are at the heart of any such description and if we don't get them correct nothing else will work out. As will be seen later, we wish to talk about an action being performed *at* (or *in*) a location and *at* (or *in*) a particular time as well as involving a certain number of agents and/or objects. We also want to be able to abstract over locations or times (e.g. to be able to talk about the landing of a plane without being specific about which landing, when or where.) Most importantly, from an information system point of view, we want to talk about what information is acquired or used associated with an action. We also want to talk about actions that occur "at the same location or time" or "at different locations and times". Finally we want to talk about orderings of actions relative to space or time. Thus we need to get straight up front a basic notion of space and time. For instance, we need to decide whether or not there are really things like "points" of space or time. We need to know just what orderings are possible and how those orderings behave. This will be the focus of this subsection.

One of the problems we face in dealing with descriptions (or specifications) [11] is that their use of the concepts of space and time often combine "point" uses with "interval" uses. For example, we speak of the foreman filling out an FDR *in 5 min.* (an interval reference) after the *login to the system* (a point reference). If this were not bad enough we then talk about the point time reference in terms of its interval perspective (e.g. the login to the system should require less than 15 sec.). Similar phenomena occur in references to space or location in space.

What we must conclude is that it is necessary to accommodate both models. Just as physicists were forced to recognize dual models of energy (particles

[11] We will use the term "description" to refer to a collection of facts about an existing situation. Loosely speaking, this collection need not be "actual", in other words it need not be complete; only "factual" (i.e. it gets correct as much as it gets). Specifications, on the other hand, are collections of facts about a possible (hopefully) or "to-be" situation.

on the one hand and continuous waves on the other), we must be able to not only recognize the duality of the space time models but also understand how to translate between them. We must also know when particular models should be used.

5.6.5 Time

All of the treatments of time we have reviewed seem to fall into one of two categories: the indexing approach, or the projection approach. Under the indexing approach each predicate or function is assigned an additional argument of type time. A primitive is introduced to represent the ordered set of all instances of type time (normally referred to as the "time line"). Under the projection approach a special function is introduced which when applied to an object and a time, returns the instance of that object at that time. We take time to be a continuous concept and model it as a combined ensemble with a set of measures and corresponding resolutions. This treatment admits to both intervals of time (pieces which are larger than our resolution) and points of time (pieces which are smaller than our resolution but not 0).

5.6.6 Generalizations and Specializations

"Mass nouns", plurals in general, and particularly plural pronouns have particularly important semantics in the processing of system descriptions. One of the most important roles they play is as labels for generalizations. Sentences like "Robocarriers have togs" delineate a part-of relationship which extends over more than just a particular robocarrier. The current situation semantics does not address the problem of generalizations, how they are formed or how they are used, yet generalizations play an important role in the description of any system. There are several subproblems associated with the concept of generalizations including:

1. Implicit and explicit quantification,

2. Types versus Classes (i.e. open or closed generalizations),

3. Definite versus descriptive reference (e.g. "robocarriers" versus "the robocarriers").

Indexing over the individuals or relations or spacial temporal locations or attributes as was done to accommodate the concepts of event types, roles and situation types does not work with this particular problem. However the extensions proposed to the concepts of properties and bindings we believe do provide a handle on the problem. From the SRM point of view a generalization is a binding without a stake. The issue of open versus closed generalization is an issue of the use (pragmatics) of such a binding in the recognition of a uniformity. If the use dictates the recognition of all of the properties or subsumed bindings then we say that the generalization represents a class. Classes can be conveniently modeled as sets. The binding can be thought of as a characteristic function (or at least the definition of the characteristic function) of that set.

If the use allows for a partial match of the properties or subsumed bindings then we say that the generalization represents a type. A type unfortunately is not conveniently modeled as a set (intentionally or extensionally). Barwise makes note of this problem in [Barwise 1984] with the introduction of the concept of "collections". He attempts to resolve this problem by definition of the concept of a "structure removing function" in [Barwise 1985b]. For the purpose of the KAMSS we do not need to resolve this issue. We suggest that it may be appropriate to use *ensemble* theory [Bunt 1985] as a basis for modeling the type concept as a quasi-continuous concept, however our work is not yet complete. Just knowing that the problem exists allows us to at least avoid the problems attendant with trying to apply set manipulation operations to something which is inherently not a set (see IDEF1/ES discussion in Section 8). From a pragmatic point of view a type is a very useful concept for initial classification of a plural reference in a sentence. We can delay the determination of "class" status until we need it (generally in resolving a conditional constraint or action reference).

Understanding "how" generalizations come about is also important for the KAMSS design, not necessarily because we want the system to be able to derive them unassisted but rather because looking closely at the process can provide insight into the representational requirements of such mechanisms. First of all we distinguish between generalizations over uniformities recognized in:

1. Physical objects,

2. Situations (or events, actions, courses of events, processes),
3. Abstract objects.

In the process of generalization there is also a distinction which can be made relative to whether the "uniformities" are based on properties which can be directly sensed, or on properties which are abstractions (models if you will, that are grounded through intermediate processes of inference). Color, size, and location based descriptions such as *the large, green parts in the west end of the plant* are examples of the first type of generalization applied to physical objects, whereas, *Long lead time items* is an example of the second type of generalization again applied to physical objects.

In all cases the formulation of the generalization requires:

1. Recognition of the relevant properties (note that this includes the attunement to the individual uniformities and the formulation of a binding of those uniformities into a unit),
2. Selection of a word to represent this binding,
3. Selection of words to name those properties,
4. Formulation of a procedural method for assessing whether or not the properties hold,
5. Axiomatization of the relationship of the generalization to other generalizations.

5.6.7 Types, Sorts, Kinds and Classes

In our ontology we distinguish between the notions of types, sorts, and classes. These notions occur frequently in dealing with descriptions, computations and knowledge representation. The tendency is to treat the terms as synonomous. However, doing so results in a total breakdown in terms of an ontology as clearly documented in [Brachman 1983; Brachman 1985; Israel 1983; Woods 1975]. Each of the terms refers to an ordering of some nature. As such they refer to means of partitioning collections of individuals. Each of the terms has differences in whether they are concept forming terms or assertional terms. That is, does their use imply extension to the ontology or to the statements about the real world made with that ontology. We will use the term **Kind** to refer to orderings of naturally occurring phenomena (i.e. the biological genus, species, phylum, b-class, order .... partitioning scheme). We will reserve the term **Type** to refer to partitioning which has spatiotemporal sensitivity. That is, it can and does change with context. We will reserve the term **Class** to refer to a partitioning which is fixed and non-overlapping. Finally we will reserve the tèrm **Sort** to refer to a partitioning which is fixed and possibly overlapping.

In general we note that the description of a characterization (e.g. class, sort, kind, etc.) has the following components:

1. Some means of referencing the characterization either by name, or description,

2. A collection of properties (definiens in the case of classes and sorts),

3. A collection of *membership* axioms which specify how something can be determined to be "of that category" (i.e. necessary conditions, sufficient conditions, and in some cases both necessary and sufficient conditions),

4. A collection of axioms of membership which specify what information can be drawn from the knowledge that something is "of that category".

The above observation without consideration for the last item could be considered to be rather pedestrian. Most of the existing knowledge engineering tools provide for specification of such descriptions in one form or another (e.g. in Frames, or Schemas, or Objects, or Structures). However, it is only relatively recently that the recognition of the problems associated with the "interpretation" of specifications made with those structures has been recognized. That is to say that the presence or absence of a concept in a type network really says nothing about the domain of interest. If we try to interpret the structures assertionally we have difficulty in formulating expressions involving incomplete knowledge. If we adopt a declarative reading we are left with little more than sophisticated data structuring mechanisms. One approach suggested by KL1 [Schmoltz and Brachman 1982] and Krypton [Brachman et al. 1983] is to recognize the create a representation scheme that provides for the distinction. In one way or another they provide a "terminology component" for supporting the construction of taxonomies of structured terms, and an "assertion component" for making statements about the real world. We feel that such an approach goes too far and hence sacrifices the efficiency of expression which natural communication depends upon. The problem with separation of the "T-box" from the "A-box" is that the world around us includes our terminology. Thus, the creation and description of a term makes assertions about the way the world is. The key to resolving the problem is not the denial of this fact but the recognition of the distinction between meaning and interpretation. Hence the focus on situation based relative semantics.

5.6.8 Situations Revisited

In the most general terms a situation is a part of reality which can be comprehended as a whole. Previously we introduced the notion of a situation type as a set of constituent sequences (i.e. $\mathcal{Y} = \langle r, x_1, \ldots, x_n \rangle$) with associated polarities (i.e. $\sigma : \{\langle\langle r, a, b\rangle, 1\rangle, \langle\langle r', a, c, \rangle, 0\rangle\}$). The introduction of the notions of properties, bindings, and stakes as the basis for recognition of and reference to uniformities does not necessarily effect a change in the S and A definition of situations. Rather they provide a finer grained interpretation of the components of that situation. Thus the "a's" (or individuals) in the above constituent sequences now become objectifications over further sets (or collections) of properties. This finer granularity provides better support for the classifications activities of utterance understanding and reasoning required in the KAMSS. It also assists in handling the notions of generalizations and actions which were absent in the original S and A theory.

5.6.9 Extensions to Conditional Constraints

A constraint is a relation between situation types σ and σ_1 such that if σ is realized then so is σ_1. Thus a "real" situation (an actual state of affairs $s \in \sigma$) contains at least the information that there exists an $s_1 \in \sigma_1$. This notion, though not explicitly stated, was the basis behind the concept of "specific relation classes" in IDEF-1 [Ramey 1983]. From the above then the *meaning* of a sentence is a *constraint*, that is, those absolute relations (which hold as long as the background conditions remain constant) between the types of situations in which the sentence is used to assert something, and the type of situation described by those assertions. The interpretation of an utterance is then a binding between the utterance and the type of situation it describes. This is true regardless of the type of the utterance. The *propositional content* of an utterance is that there *is* a situation of the type it describes. The "meaning" of a sentence then is a relation between the type of situation in which the sentence is used to assert something, and the type of situation described by the utterance. A conditional constraint then is a constraint type with an associated definition of the conditions under which that constraint type is effective. The specification of a conditional constraint then involves the specification of the background conditions, the participating situation types and the absolute relations which hold to convey a specific meaning between those situations.

If–Then Structures:
If – Then structures in utterances provide a mechanism for stating *the conditions* for a conditional constraint. The utterance in the antecedent portion of such a construct describes a situation or set of situations which at least existentially entails the situation included in the consequent portion of the construct. In our analysis of the manufacturing texts (see Appendix A) the if–then structures appeared to be used only when the author was attempting to describe universal constraints, or a specific mechanism of which he had detailed knowledge.

When–Then Structures:
When – Then structures in utterance provide a mechanism for stating *evidence of* conditions for a conditional constraint. Generally by use of reference to a staked perception the "when" portion of the utterance describes a situation which was observed to stand in a constraining role with the situation described in the "then" portion of the utterance. The typical use of these structures is associated with co-occurrence of information about two situations. The "when" conditional is used as a manner of reporting this co-occurrence without establishing a strong cause/effect relationship statement.

5.6.10 Accounting for Change

Accounting for actions first requires defining at some level:

1. What an action is (something perceived or something done with a perception).

2. How actions relate to the elements previously defined.

3. What is the relationship of beliefs and intentions to actions.

4. How do actions individuate (e.g. Is moving the autobody a part of finishing the car.)

5. What is the relationship between information/knowledge possession and actions.

According to Hass one knows what an object is if one knows enough about it to carry out one's intended actions [Hass 1986]. In the robot world you need knowledge to perform an action when you must find out which commands will produce the desired actions. From the point of view of the KAMSS there are two different kinds of actions:

1. Actions it is being told about (e.g. A machinist sets up the framing fixture).

2. Actions which it knows how to perform (e.g. print a message to the screen, or design a simulation model).

The first variety is important since actions and their descriptions are central to the description of the manufacturing system and to the reasoning about how it works. The second variety is important because KAMSS must know how to respond to specific commands or situations. For example it must know when it has recognized an instance of the first variety and produce the appropriate data structures and linkages for its storage.

The original situation theory does not provide a direct way of describing actions. The basic concept of a situation is defined in terms of the objects and properties which hold at some spatiotemporal location. To give an accurate account of the notion of action we must take on directly the problem of describing *change.* We believe that this requires some basic modifications to the situation theory. Essentially what we propose is to recognize states of affairs as being different from what we will call *states of change.* The basic motivation behind the notion of a state of change is the ability to describe a situation in which some of the objects, properties, or relations participating in that situation are recognized as being in the process of changing. Rather than introduce another completely new syntactic structure for denotation of states of change we propose the extension of the situation type syntax to extend the "0, 1, undefined" indicator set to include the indicator "changing". Thus, unlike most other AI ontology treatments of change, we allow for the description of:

1. What is changing?

2. What is remaining the same?

Even with this addition there are three other important elements of a description of change which we have to add to the situation theory. Since the descriptions are formulated at the type level there is the problem of describing sequencing and timing. There is also the problem of describing aspects of causality, enablement, coupling. Finally there is the need to describe characteristics of the "realization" of a type. To address these ontology issues we must extend our own treatment of change description to include realization constraints to the situation description. The situation theory presented above provides most of the tools we need to do this. For example, if we want to represent things that change over all instances of a state of change type we use indexicals to denote those things. If we want to represent things that remain the same over all instances we use constants to denote those things. Since we have at our disposal the ensemble theory we can formulate constraints over the indexicals, their anchoring functions (which gives us access to the instances) and (via the SRM primitive set) to the possible interpretations of the abstract objects in the description (e.g. constraints on the values of a measure associated with time interval of a state of change). Thus we can describe constraints on the co-occurrence of two states of change which are of the same type (e.g. cabs arrive every fifty seconds). The following sections describe structures in our ontology which address the issues identified.

5.6.10.1 Processes

The situation theory notion of an event and courses of events described in Section 5.4 are preserved unchanged. However the addition of the distinguished changing elements of a situation description allows us to build into our ontology the notions of *processes* and *actions*. These notions are critical to the capture of descriptions of the way that a manufacturing system works. *Processes* become interpreted as ordered sequences of courses of events, states of affairs and states of change. This corresponds nicely with the common sense notion of a process being a set of related events spaced out in time with intervening steady states. Processes provide the framework for describing a class of relations that can hold between instances of states of change types. Using the same mechanisms described above for a single state of affairs or state of change we can for example specify constraints such as:

1. Temporal constraints between states of affair types or changes (e.g. that an event a of type A occurs some time before event b of type B).

2. The end of one state of affairs or state of change is synchronous with the beginning of two or more states of affairs or change.

3. The end of two states of affairs or change are concurrent.

4. Inertial delay (i.e. the fact that a process can be in the state of occurring for some finite time before we can detect that the properties which are supposed to be changing are changing.

5. Cycles (repetition, iteration, or looping) of states of affairs or change.

5.6.10.2 Actions

Actions, on the other hand, are commonly thought of as states of change which involve at least one causal-agent role type. That is, an action is an event which is caused by the agent. This characterization ties the notion of causality into the definition of a situation type which we believe to be basically incorrect, because it masks the more subtle types of causality (e.g. enablement, inhibits, coupling). We choose to interpret actions as indexed constraint types involving at least one state of affairs and one state of change with the common indeterminate (or set of indeterminates) standing in the agent role relative to the participating state of change type. Thus actions become a special class of causality contraint types. This characterization of actions is very useful in the modeling and reasoning about how a system works (as discussed in the next two sections) because it allows us to postulate from a state of affairs how a state of change comes into being (or vice versa) by postulation of the existence of the missing component in an action indexed constraint type. This gives us a built in version of a kind of

Newtonian law of momentum (states of affairs tend to stay states of affairs until acted upon ...).

5.6.11 Model Ontology

Situation theory takes the individuals, relations, and spatiotemporal locations as primitives for building complex objects which can be used to classify reality. For descriptive purposes types are introduced, and for representational purposes abstract versions of both the primitives and the types are introduced. Our intension is to use the typed abstract situations to construct descriptions of systems. However, we also have the need to generate and manipulate interpretations of these descriptions. That is, we need to construct models of our descriptions. These models will be used to support such activities as qualitative (and quantitative) simulation as well as data acquisition planning. This model world (or Token View) has similarities to both the real world and the abstractions of that world captured in our descriptions. It can be thought of as an abstract world without types. It consists of real symbols (in our case stored and manipulated within the KAMSS) these symbols are the objects of this world. In Section 7 we will discuss how we generate these token worlds in order to provide a basis for certain types of reasoning based on constraint propagation and propositional logic.

5.7 Representation Structures in KAMSS

As mentioned previously in this section there are several interrelated but orthogonal factors driving the issue of a computerized representation for KAMSS. Those issues are: retrieval, reasoning, and computation. In this section we will describe a representation approach which attempts to accommodate the needs of all three issues. As discussed in Section 3 during the course of this research we reviewed a number of representation schemes including:

1. KL0, KL1, and KLONE,

2. KRL,

3. Units,

4. LOOPS [Stefik et al. 1983],

5. First order predicate logic (e.g. Prolog and ESP),
6. SRL / CRL,
7. ART schemas and rules,
8. KEE [IntelliCorp 1985a],
9. Flavors,
10. NETL [Fahlman 1985],
11. Syntactic Theory (named expressions),
12. POP-11,
13. FOOPLog,
14. Portable Common Loops [Stefik and Bowbrow 1986].

We also implemented a type based system as a part of the SDC prototype. The purpose for the review was to learn what others had attempted in the design of a programming language for representing descriptions, models, and of course programs. It should be noted that the design of a representation is the design of a programming language. Experience dictates that such a task is not trivial. Therefore, in the sections which follow we have attempted to circumscribe the major features of the language we have in mind, leaving the details for future work.

5.7.1 Language Requirements

The languages reviewed as a part of this research exhibited several basic flaws. One of which is the lack of a formal semantics for interpretation. We believe that the semantics should be developed prior to a commitment to syntax. The reason being that this generally results in a cleaner syntax and a simpler underlying implementation. The existing languages also do not cleanly support the representation needs of the three usage paradigms noted above. Most of them exhibit a primary focus (e.g. support of computation) and address the other paradigms as an after thought. For example, consider the role of a schema in CRL or ART.

While the initial intent of the "frame" construct was to characterize a situation, the schema construct in these languages serves primarily as a data structuring mechanism [Fikes and Kehler 1985]. Similarly "objects" in LOOPS or Units or KEE have a primary role as program modularization constructs. They serve as excellent vehicles for storing state information about the computation, and for defining a uniform protocol for computational behavior. However, they do not provide a convenient mechanism for capturing descriptiòns representing the semantics presented earlier in this section[12]. Nor do they support the construction of a system which has a capability to formulate different semantics as a natural part of its processing of descriptive input. Finally the representation of knowledge in these structures is not in a form convenient for recognition based reasoning which will be described in Section 6. It is not the case that these languages are not useful, in fact we would argue that they serve as an excellent basis for implementation of the language we have in mind.

The logic based languages (such as Prolog [Clocksin and Mellish 1981] and its object extension ESP) in their desire to be efficient computational devices have sacrificed the representational flexibility of first order predicate logic. At the same time they suffer from the inability to conveniently handle such programming necessities as screen and file I/O and variable side effects (since logical variables can be unbound by backtracking)[Kowalski 1979, 1981a, b].

The rule based languages fail to provide a convenient mechanism for representation of axiomatic knowledge associated with a concept (a critical need for direct representation of an ontology). These languages also suffer from the problem (which is shared by the logic based languages as well) of implementing a specific reasoning method, without the programming power to be usable to implement variants of that method.

What is needed is a programming language (with the associated implementation) which:

---

[12] It is interesting to note that Flavors, VBASE, Smalltalk and C^{++} make no such knowledge representation claims. They are simply offered as object oriented programming languages, promoted as tools for constructing modular, reliable, and reusable code.

1. Offers the computation specification clarity of a functional language.
2. Offers the modularity of an object oriented language.
3. Supports the development and specification of an ontology.
4. Supports the application of that ontology to a particular domain.
5. Supports the construction of reasoning mechanisms which can both use the provided ontology and/or extend it.
6. Provides for temporary and permanent storage of data generated and used.
7. Provides for the management of evolving systems descriptions.

Attempting to satisfy all of these needs turns out to be somewhat more complex than would first appear. As Wirth has pointed out the elegance of a language is more determined by what you leave out than by what you put in[13]. Following this theme we would want to strive for as lean a language as possible. There are also conflicts to deal with in the design goals which have been proposed for the various languages which have been designed to address the individual needs stated above. For example, in programming languages for computation specification an oft declared goal is for "referential transparency" (i.e. contextual independence of interpretation). On the other hand the basic premise behind the SA ontology presented in the first part of this section is that one of the central properties of a useful language for communication is contextual *dependency*. These goals are not necessarily mutually exclusive, but it is certainly the case that one can easily design a language which satisfies the first goal and is incomprehensible (and hence difficult or awkward to use). The point is that an attempt to patch together existing language constructs into a new language without careful consideration of these various tradeoffs would probably result in a system lacking any intellectual coherence.

[13] This comment is generally made in reference to PL1 or Ada

5.8 Another Notational Anomaly (ANNA)

The following section describes the basic structure of a language for expressing an ontology, and the application of that ontology based on the structures presented above.

5.8.1 Symbols, Gramatical use, Interpretation use, and Interpretations

The first issue to deal with in the design of the ANNA language is the representational distinction of the symbol, from the symbol use, from the interpretation of the symbol. In ANNA we distinguish the symbol by use of the LISP like **quote** notation. Thus, 'ANNA denotes the symbol consisting of the letters "A", followed by "N" and so forth. We distinguish two types of symbol use: grammatical and interpretable. The first is related to how a symbol can be combined with other symbols to support the second kind of use. Thus, 'conveyor denotes a symbol but '(the conveyor) denotes a grammatical use. The distinction between grammatical use and the symbol classifications is that a symbol does not necessarily have an interpretation, whereas, a grammatical use will always have an interpretation. Now, of course, there are single symbols with intepretations (e.g. proper names being a large class) and collections of symbols without an interpretation (e.g. dangling participial phrases). The interpretable use of a symbol or set of symbols is denoted by enclosing the symbol in square brackets. [ANNA] denotes the use of the symbol 'ANNA and [the west end loader] denotes the use of the set of symbols 'the, 'west 'end, 'loader. The interpretation of a use of a symbol is denoted by the LISP like **backquote** notation. Thus, the interpretation of the use of the sentence '(Anna likes Celeste) is denoted '(,[Anna likes Celeste]). This notation allows us to reverse the traditional notion of constants and variables[14]. It also allows the embedding of referents since the evaluation of a grammatical use is recursively applied to the grammatical use of each of its constituents. Thus '(,['(,[Anna]) 'likes '(,[Celeste])]) is the first level of evaluation of the previous interpretation of the use of the sentence '(Anna likes Celeste). Notice that the symbol 'likes does not receive an interpretation this is because it has no grammatical

[14] That is variables, and not constants, are used to stand for arbitrary objects.

use as a stand alone symbol. It will be used in the construction of the interpretation of the overall sentence. A more complex example might be as follows:

```
‘(,[The west end loader moves autobodies from the BIW area to the Uniprime
conveyor])
->
‘(,[ ‘(,[the west end loader]) ’moves ‘(,[autobodies]) ’from ‘(,[the BIW area])
’to ‘([the uniprime conveyor])])
```

Bindings

Since by our previously defined ontology we wish the interpretation of a grammatical use of symbol(s) to be a characterization of the situation(s) of the type so described by those symbols and since a situation is described by the facts it supports we need a way of denoting **facts**. Facts can be thought of as asserted constituent sequences with an associated polarity. Thus a fact description is a binding with at least a relation name and a polarity specification. Optionally (and as is the more common situation) fact descriptions include a set of argument names with associated values. We choose a schema like representation of a fact with the following form:

```
(binding
:name <symbol> ; this is optional
:relation-name <a symbol> ; required
:polarity <0 or 1 or undefined or changing> ; required
:<argument-name> <argument value> ; zero-one-or-many of these
.
.
.)
```

Variables

In order to use facts to classify, type, and sort situations we must allow for any of the constituents of a fact to have as its value a variable or unspecified value. This is represented by the use of the keyword :var in the parameter value position of the fact schema. The :var keyword can be optionally followed by a symbol which denotes the name of the variable value.

```
(binding

:name <symbol> ; this is optional

:relation-name <a symbol> ; required

:polarity <0 or 1 or unknown> ; required

:<argument-name> :var X

.

.

.)
```

Note that this applies to all of the parameters including the relation-name as follows:

```
(binding

:name <symbol> ; this is optional

:relation-name :var X ; required

:polarity <0 or 1 or unknown> ; required

:<argument-name> :var Y

.

.

.)
```

Anchors

In the section on situation semantics we introduced the notion of functions which assign values to indeterminates in constructing events from an event type. More generally, we want to be able to represent restrictions on what can be the value of a variablized parameter of a fact. The existence of an anchor is denoted by a '—' following the variable name or the :var designator in the case of an un-named variable. The anchor itself is a function of two types - either a constructor (e.g. a function which can create or find an appropriate filler for the variable) or a restrictor (e.g. a function which can determine if a proposed value of a variable is appropriate). The type of anchor is designated immediately following the '—' with a :con or a :res. A variable may have either or both types of anchors. The relative expressiveness (or power) of the language provided by the representation system will be determined partially by the set of built in anchoring mechanisms provided for the various elements in the ontology. For example, does the implementation provide:

1. basic set manipulation operators,
2. ensemble operators,
3. concept subsumption operators,
4. classifiers,
5. type manipulation operators,
6. sort enforcers (better known as data type enforcers).

In fact it is the definition of the language for anchor specifications which requires addressing the other computation and data access/manipulation features of the language requirements stated above.

Situations

A situation is described by a binding over facts and stakes. Internal to the system the situation is actually defined or classified by use of the binding schema with the following distinguished parameters:

1. :name optional,
2. :in-s to denote that the schema classifies a situation as far as it goes,
3. :—= to denote that the schema is meant to define a situation,
4. :stake-*i* to denote a stake constituent which may have a value or a variable with associated :con or :res anchors,
5. :-> to denote that the situation described is informative on another situation given either by *name reference* or explicitly spelled out.
6. Fact-*i* one-or-more facts either by name reference or explicitly spelled out.

Note that one of the interesting things we can do with such a representation is to easily coerce a fact into a situation descriptor merely by the addition of a header declaring it as such. Similarly the facts that by association describe a situation are easily extracted. This capability will be important in the later sections on reasoning and modeling. Also note that the situation description can be extended arbitrarily through the addition of more facts.

All of the ontology elements introduced previously can be described using only the above defined language syntax. The following is an illustration of several of the more interesting items.

Conditionals

The "When" and "If" type conditionals are handled as relations between situations. Thus, for example, the interpretation of the use of the sentence:

```
'(When the forks retract the cab carrier is released)
```

would be represented as

```
'(,[When the forks retract the cab carrier is released])

-->

'(,['(,[When the forks retract]) '(,[the cab carrier is
                                            released])])

-->
```

```
  ‘(,[‘(,[’when ‘(,[the forks]) ’retract])
    ‘(,[‘([the cab carrier]) ’is ’released]) ])
```

The interpretation would be described by a relation between the two referenced situations, the one described by the forks retracting (i.e. *S*) and the one described by the cab carrier being released (i.e. *S'*) as follows:

```
(binding
        :name release-trigger
        :relation-name involves
        :polarity 1
        :enablement  S
        :effect S’
        )
```

This seems to nicely capture the natural semantics interpretation of 'When as a reporting of a co-occurrence of sets of facts (i.e. just the facts describing the two situations).

Whereas the interpretation of the use of the sentence:

```
  ’(If the forks retract the cab carrier is released)
```

would be represented as

```
  ‘(,[If the forks retract the cab carrier is released])

  -->

  ‘(,[‘(,[If the forks retract]) ‘(,[the cab carrier
                                       is released])])

  -->

  ‘(,[‘(,[’if ‘(,[the forks]) ’retract])
          ‘(,[‘([the cab carrier]) ’is ’released]) ])
```

The interpretation would be described as an informative parameter in the situation described by the forks retracting (i.e. *S*) having as its value the situation described by the cab carrier being released (i.e. *S'*) as follows:

```
(binding
        :name S
        :in-s
        :polarity 1
        :stake :var L  :res (spatiotemporal L 1)
:-$>$ S'
)
```

Which seems to capture the corresponding stronger notion of the "IF" conditional.

5.9 Summary

Using the above syntactic approach a description of a situation becomes merely a collection of facts. There is no restriction on the type or number of these facts. We will show in the next section how we can capitalize on this characteristic of a description representation in order to provide a basis for the reasoning capabilities required by KAMSS.

In summary, this section presented:

1. The need for a formal semantics for design of the representation systems in KAMSS.
2. An introduction to situation semantics as a basis for the ontology behind description and model representation in the SDCE and MDSE components of KAMSS.
3. A method for semantic theory evolution.
4. The conceptual structure for extensions to the situation semantics basis to account for continuous concepts.
5. A language structure for representation of descriptions which can accommodate both acquisition and reasoning processes using such descriptions.

6. REASONING IN KAMSS

The main purpose of this section is to present:

1. The criteria which a reasoning method must satisfy to be considered robust enough for those tasks described in Section 5.

2. The types of reasoning which needs to be supported in a system such as KAMSS.

3. A hypothesis that a rational approach to modeling human reasoning situations in the context of the activities set out in Section 2 and the approach to semantics proposed in Section 5 is to separate reasoning into a discovery component and a method component. With this separation we argue for an approach to reasoning which is based on "chains" of information flow.

6.1 Observations on Reasoning in KAMSS

The terms "reasoning, thought, argument, proof" all characterize intellectual tasks. Theories for these tasks are attempts to establish methods for the application or structuring of these tasks so that the result of application has certain properties. In its most basic characterization reasoning is an activity through which an agent attempts to use known facts about the world in order to extract additional information. This additional information is assumed to be implicit in the facts already in the possession of the cognitive agent. Thus, the concepts presented in Section 5 of meaning and information are central to the process of reasoning. For the way that we come to know information, and the way we extract, store, and manipulate information is central to the manner in which we can use that information in a reasoning process.

Reflecting on the types of cognitive tasks described in Section 2 it would appear incorrect to assume that a person employs a particular reasoning method. Rather it is more reasonable to assume that the customer and the analyst have the capability to generate a form of reasoning method that is appropriate to the task at hand. It is reasonable to argue the fact that the human has the capability to change reasoning methods based on the representation which is provided by the theory of semantics that he is employing in a given situation. Or that the human can alternatively change his representation to suit the method of reasoning which he would like to employ. What this implies is a view of reasoning as containing a discovery component and a method component. What we would propose is that the discovery mechanisms of reasoning are more basic than the method mechanisms. The method mechanisms we will refer to as the "logic" component (see Figure 6.1).

We propose that for a theory of reasoning to be acceptable it should have the following properties:

1. It must be shown to be the basis for a discovery mechanism for language acquisition.

2. It must be shown to be a discovery mechanism for the models which have heretofore been postulated as the basis for reasoning (i.e. categorical logic, symbolic logic, propositional and predicate calculus, and non-monotonic logic, as well as inductive reasoning systems.)

3. It must be shown to be simple enough to be acquired by an individual even in the context of a relatively "dirty" (non-monotonic, irrational) environment.

The remainder of this section outlines an approach to a theory of reasoning which we feel could be developed to achieve the above defined goals. The following sections will sketch the basic concepts of information flow as the basic mechanism for support of the reasoning requirements in KAMSS, describe how this concept is consistent with the ontology structures presented in the previous section, and illustrate how this concept would work to support the basic knowledge acquisition process. In the next section we will discuss how the concept supports the model design and situation analysis process.

6.2 Types of Reasoning in KAMSS

Considering the cognitive tasks outlined in Section 2 and 5 the reasoning skills required of the KAMSS would include but not be limited to:

1. Reasoning with defaults,

Type 1 -	Reasoning required to acquire system description
Type 2: -	Reasoning required to communicate what KAMSS knows.
Type 3 -	Reasoning requied to determine missing knowledge needed to answer a question.
Type 4 -	Reasoning required to answer a question.
Type 5 -	Reasoning required to design a model to answer a question / explain a behavior.
Type 6 -	Reasoning required to analyze a model
Type 7 -	Reasoning required to design a solution to a problem.

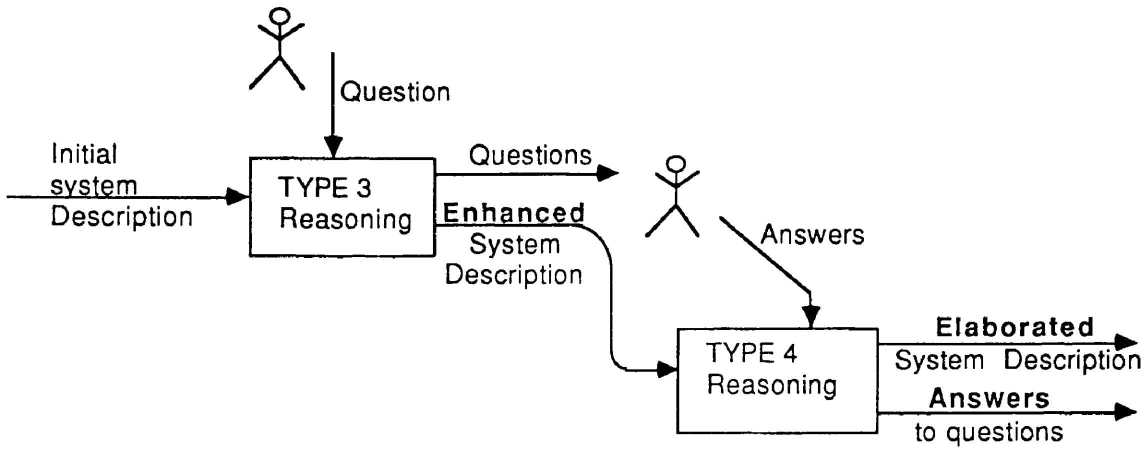

Question
Initial system Description
TYPE 3 Reasoning
Questions
Enhanced System Description
Answers
TYPE 4 Reasoning
Elaborated System Description
Answers to questions

FIGURE 6.1: REASONING COMPONENTS.

2. Reasoning with time intervals and time points,
3. Reasoning with situations,
4. Reasoning with knowledge and beliefs,
5. Reasoning about cause and effect relations,
6. Simulation of the reasoning of another agent (i.e. the user),
7. Goal extraction from situation understanding,
8. Plan construction ,
9. Experiment / test design,
10. Plan failure analysis,
11. Test result interpretation.

An important point which can be made about the reasoning in KAMSS is that it is reasoning about things that it has been told. It is not envisioned in the current prototypes to have any means of acquiring perceptions of the outside world beyond the "utterances" which come in to it via the terminal. This need not be the case. For example the KAMSS could be fitted with an interface to a factory information system, or a product / facility CAD data base. Under such circumstances the system could collect something of its own observations. As illustrated in Section 5 each utterance will have at least the following kinds of information associated with it:

1. A classification (i.e. command, declaration, conditional, drawing, etc.),
2. A time stamp (generated by KAMSS),
3. A person ID, a unique identifier of the person at the terminal generating the utterance.

There are two main classes of reasoning of issue in KAMSS. One class is referred to as "data driven" (i.e. inference performed on data acquired). The other class is referred to as "goal driven" (i.e. inference performed on demand). The former is primarily triggered during the discourse acquisition process. The latter is primarily triggered via queries or commands posed by the user. In the previous section we identified two types of queries. Those that request data about facts or of facts acquired, and those that query about possible situations. In the latter sections of this section we will see that these two types of queries pose very different problems for the reasoning system in KAMSS. But first we will address the problem of data driven inference.

6.3 Data Driven Inference

One issue that we must address is just what does KAMSS "know"[1] on the receipt of an utterance like "The Dodge City wheel and tire line delivers wheels/tires to the main assembly line." We suggest that there are at least two classes of information which can be acquired from the processing of such an utterance. The first class is referred to as the semantic meaning, the second the "pragmatics" meaning. Given that the KAMSS can classify the components of the sentence into semantic case structures or some internal representation as was illustrated in Section 5, we claim it has access to the semantic information from that utterance. However, the pragmatic information will turn out to be the more useful component in the general case. Pragmatic information content is that derivable from the discouse situation and the assumptions of the semantic theory upon which the KAMSS reasoning system is constructed. Presuming KAMSS could parse this sentence and determine that it is an English sentence, since the utterance would have been received from the current user of the system, one of the "things" that KAMSS knows is that that person believes that a situation (of the type described by that sentence) exists.

[1] As our intent is to focus on the characterization of the representation of the ontology and the principles of reasoning need to support a KAMSS we will side step the issue of the problems attendant with the fact that axiomitization of "to know" typically results in the need to introduce a truth predicate which (using the results of the fixed-point theorum) gives rise to all of the classic incompleteness results. In this section we will use the term "know" in the sense of "believe" which does not carry with it the truth predicate requirement.

KAMSS should also know that the information content of the statement made with that sentence is dependent on the types of situations in which such a statement can be used and upon the conditions underwhich information is communicated in those discourse situtations. KAMSS also knows that those conditions can be parametric or parameter free. That is KAMSS knows that some constraints which carry meaning will have free variables which are normally fixed by the context of the discourse (parametric) whereas others contain no variables at all. We propose in the following discussions that the reasoning process uses parametric constaints picked up through the discourse process (or preprogrammed as a part of the domain knowledge) as axioms to guide the discovery mechanisms for hypothesizing a situation reference which is needed to complete a reasoning link. We propose that the parameter free type constraints (preprogrammed or acquired) provide the basis for generalization when the situations are known and the constraints are being sought.

From a mechanistic point of view all that KAMSS "knows" is represented by symbols in the KAMSS knowledge base (we will avoid the meta-physical arguement of whether the same is true for humans). References to situations actual or hypothesized are either explicitly labeled or unknown to KAMSS (we do not propose that KAMSS knows what it *can* reason to, only what it has acquired or has reasoned to). Thus what we need from a mechanism point of view is a syntactic manipulation approach which can accommodate the types of discovery and method components we have in mind. This leads us to the consideration of a combination of the "syntactic theory of belief and action" of Hass [Hass 1986] and the "description matching" framework of Bobrow [Bobrow and Winograd 1977] can be used as a basis for a reasoning method for the KAMSS.

One of the sticky issues associated with the KAMSS reasoning system is that of how to deal with various types of assumptions, particularly those associated with assumptions about the beliefs of the user. For example, suppose the user "knows" that KAMSS "knows" a particular fact. The KAMSS reasoning system should be able to determine that an ambiguous reference (made by the user because of his belief) can be satisfied with that "known" fact. One possible basis for resolving such a problem is to insist that the some basic principles of reasoning about knowledge be adhered to. The first would be the principle that, if someone (including the KAMSS) knows some thing then he knows that he knows it [Moore 1980]. This would allow the KAMSS to attempt to resolve the ambiguity by inferring the missing information from its knowledge base to resolve the ambiguity and then asserting the consequences as an assumption. An example of this sort of reasoning applied to the construction of the meaning of an utterance is provided in the following section.

6.3.1 Construction of the Meaning of an Utterance

One of the problems with theories of semantics is that in order to be general enough to handle the problems of explaining the efficiency of language and the capabilities of humans to freely objectify concepts they tend to be too finely grained for actual implementation use. The reverse problem is that semantic structures which have been established for particular applications tend to be too course grained and brittle for reliable use. What we feel is needed is a hierarchy of conceptual structures, and a flexible means of moving from level to level. The following section illustrates the concept we have in mind by example.

An example of the strategy of use of the SRM in the context of utterance meaning construction in KAMSS is presented in the following. Suppose that KAMSS receives an utterance of the following:

1. The robocarrier moves autobodies from receiving to paint.

Let us assume for the moment that the domain knowledge of KAMSS has no information related to the symbol string "robocarrier". KAMSS knows that the interpretation of an utterance is the type of situtation it describes and that the propositional content of declarative sentences is that there is a situation of that type. KAMSS also "knows" something about the language of the user. That is, we can assume that words like "the, a, to, from, after, before, when, if, ..." are recognized as a part of the base language, and that the semantics of these words have been worked out by the developers of KAMSS. We can also assume (see Section 4) that the parser can recognize that the sentence is a well formed English sentence, and that for example the string "moves" is the main verb of that sentence. The issue here is what to do with the strings "robocarrier", "autobodies", "receiving", and "paint". First of all KAMSS must characterize the situation in which this utterance has been used. That is, KAMSS knows that it is in the system description capture mode. It also knows that the input which has been typed in was typed in by a user (lets call him Jon). Now the following represents the knowledge base of KAMSS at this point.

1. (utterance#1, says 'Jon "The robocarrier moves autobodies from receiving to paint") ;from the input event
2. (believe utterance#1 is a sentence) ;from the output of the parser
3. (believe there exists a situation type S1 described by utterance#1) ; from situation theory.
4. (believe there exists a situation s1 of type S1); from the propositional content of utterance #1.
5. (believe "robocarrier" binding); If you don't know what something is assume it is a binding.
6. (believe "autobodies" binding); If you don't know what something is assume it is a binding.
7. (believe "receiving" stake); From parser's classification based on case marker "from".
8. (believe "paint" stake); From parser's classification based on case marker "to".

Now suppose the next utterance which the KAMSS receives is:

1. The robocarrier moves at 20 feet per minute.

This is a property observation statement. The knowledge base of KAMSS at this point would be augmented to include:

1. (believe utterance#2 is a sentence) ;from the output of the parser
2. (believe there exists a situation type S2 described by utterance#2) ; from situation theory.
3. (believe there exists a situation s2 of type S2); from the propositional content of utterance #2.

4. (believe same sl s2); From the co-occurence of the string "robocarrier" and the use of the definite descriptor "the".

5. (believe property ?x "20 feet per minute" in binding "robocarrier");

KAMSS knows how the property is described and its value but it doesn't know what it is called.

Now suppose that the next utterance to be processed is:

1. Autobodies are spaced 10 feet apart on the robocarrier.

The knowledge base of KAMSS at this point would be augmented to include:

1. (believe utterance#3 is a sentence) ;from the output of the parser

2. (believe there exists a situation type S3 described by utterance#3) ; from situation theory.

3. (believe there exists a situation s3 of type S3); from the propositional content of utterance #2.

4. (believe different sl s3); From the plural reference and the lack of use of the definite descriptor "the".

In the manner illustrated above the KAMSS would be able to build up a description of the objects in the system, the properties associated with those objects, the situations in which those objects participate and the relations they participate in as a part of those situations.

6.4 Adaptive Reasoning, A Frame of Reference Based Reasoning Method

The basic notion behind our adaptive reasoning AR concept is that human reasoning is limited, largely pattern based, and primarily memory resident. One of the reasons behind considering the AR approach to be a viable concept is the way humans use symbol tools to extend their reasoning capabilities. It has been suggested by Harman and Hass that beliefs and knowledge are expressions of a representation language (i.e., "the medium is the message"). If we assume a fairly limited reasoning mechanism which is constrained by the characteristics of the mechnaism on which it is implemented, then one of the reasons behind the development of symbol tools (e.g., language, song, text, figures, etc.) could be that they serve as mechanisms for extending our capabilities to reason about complex situations. Thus the fact that almost every formal proof of a logic or mathematical construct involves the setting out of a notational system is possibly evidence of the need for the notational devices to formulate the constructs in the first place. Stoy points out [Stoy 1985] the effect of the Arabic numbering systems for denotation of numerals at simplifying the process of performing simple arithmetic operations from one of complex reasoning to one of simple syntax manipulation. We contend that symbolic and syntactic structures represent a tool making capability which humans are constantly using to extend rather limited reasoning capabilities to impressive heights.

Considering the limited success of the techniques reviewed at modeling human reasoning systems, and knowing that the operational requirements of the KAMSS system all but preclude the "brittle" approaches of traditional reasoning systems, it would seem reasonable to consider taking this alternative approach. After all, humans presumably began reasoning eons before the first formalizations of Aristotle's categorical methods emerged. Similarly we can directly observe reasoning, understanding and problem solving in lower forms of intelligence than man. As Harman points out, "Clearly, argument or proof is not at all the same sort of thing as reasoning in the sense of reasoned change of view. There is a clear difference in category. Rules of argument are principles of implication, saying that propositions (or statements) of such and such a sort imply propositions (or statements) of such and such other sort." [Harman 1986].

What we are essentially proposing is reasoning as a discovery process. This discovery process is one of establishing chains, the links of which allow information to be acquired from the environment, or from previous information acquired as illustrated in Figure 6.2. The AR model of reasoning is a localized model. Each chunk of information is assumed to have a logic of its own. This logic mechanism is responsible for the following primitive activites:

1. Determination of when the chunk of knowledge can be used in a reasoning chain.

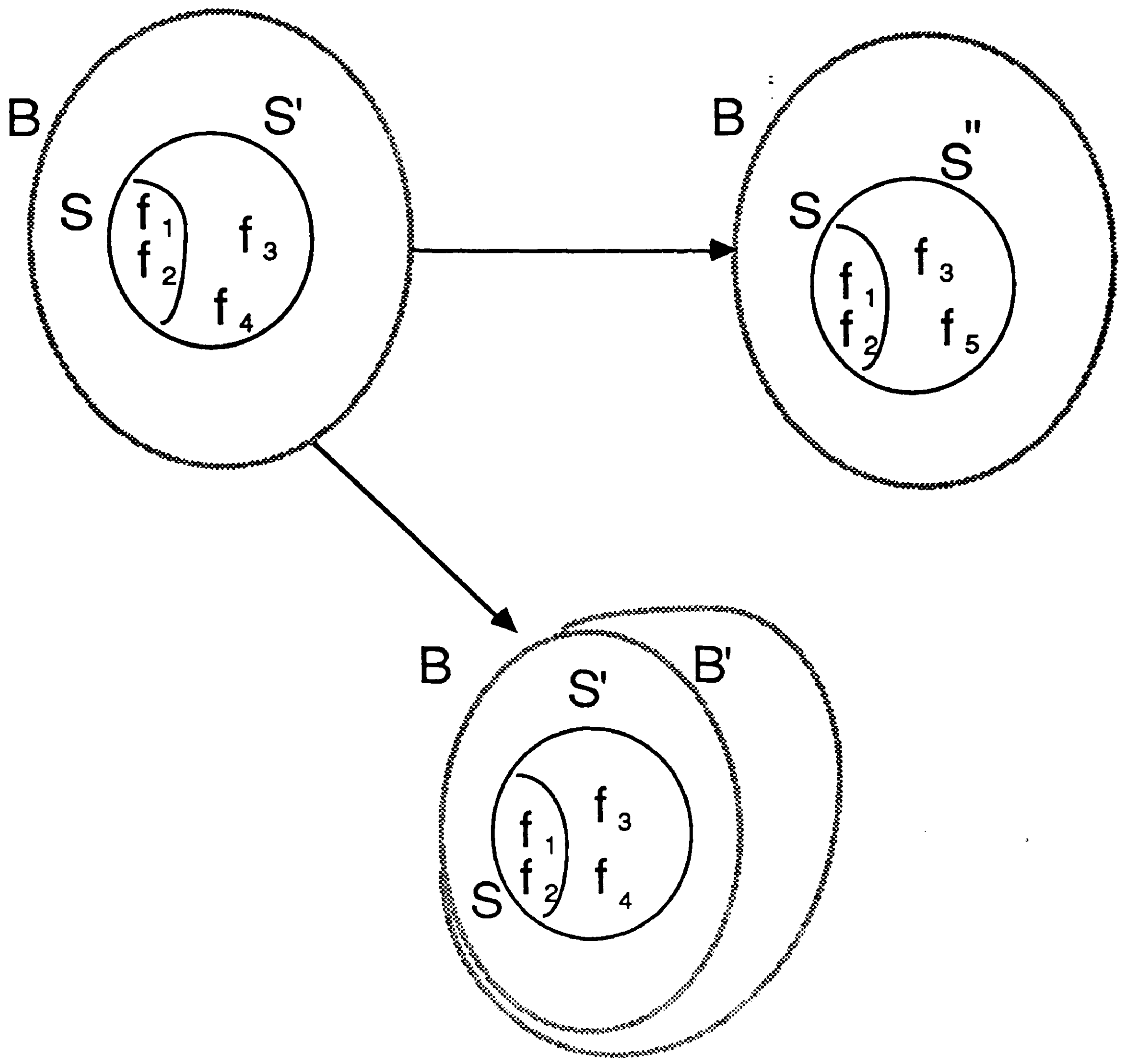


FIGURE 6.2: REASONING MODELED AS CHAINS OF INFORMATION FLOW.

2. Determination of what part of the information chunk will be passed on based on the reasoning situation.

3. Determination of what effects inconsistencies in the reasoning situation and the information use requirements will have.

The key questions to be addressed in order to characterize AR include:

1. What is in an information node?

2. What is in a link (i.e., what form are the constraints)?

3. What is the logic model which could operate at each node in an independent fashion and produce information chains which stretch across many nodes.

6.4.1 Formalizing the AR Theory

One of the first requirements of the AR approach is that it must support the SRM theory of semantics presented in Section 5. In fact the two are closely intertwined in that we will show that AR derives its feasibility from the existence of SRM. And that for the SRM to be a useful concept requires the availability of some sort of AR mechanism. From the example at the end of Section 5 we are already familiar with the following principles of information flow in a discourse situation:

1. The "interpretation" of an utterance is the "type" of situation it describes.

2. The "propositional content" of an utterance is that there is a real situation of that type.

3. The "meaning" of a sentence is the "relation" between the "type" of situation in which the sentence is used to assert something, and the "type" of situation described when the sentence is used.

4. "Meaning" consists in constraints between types of situations, and it is such constraints that allow a situation to contain information.

5. Thus we distinguish between the "meaning" of a declarative sentence, and the "interpretation of the utterance" of that sentence.

6. "Statements" are certain kinds of utterances made with declarative sentences.

Thus we do not suppose that sentences in isolation are either true or false. Rather the propositional content of an utterance is the carrier of the traditional truth value. A key ingredient in understanding the information content of a "sentence" is understanding the subject matter of statements made with it. Thus understanding a sentence is understanding what situations it can describe and under what conditions it can describe them. But not every thing about a situation is necessarily stated. Barwise [Barwise 1985a] introduced the Principle of "exploitation of environmental constraints" (What stays fixed does not need to be made explicit) to accommodate this problem.

The notion of constraints between situation types as the basis for information content of a situation can be used to define the information nodes in our chains. A node is simply a situation description which can be recalled (a uniformity which the reasoning process has access to). A link is established between two such nodes when one situation has been observed as having a binding to another situation (possibly via an statement use in a processed utterance). The simplest type of link is an existence link. That is the meaning of the first node is that when an instance of that type exists so does an instance of the second type. A more general type of link between two nodes is the "informative" link. Specifically a node is informatively bound to another node if it is the case that the facts which form the description of the first node are a subset of the facts which form the description of the second node and if the descriptions of each node are *actual.* Thus if KAMSS is aware of an informative link between two nodes and the description represented by those nodes contain variables then the informative link tells how those variables can be bound. Now as a description can also contain stakes and variables in the facts which form the description of the situtation we have a way to coerce groundings of constraints so represented.

Part of the "discovery" process of reasoning then is the "noticing" of the constraints between the various types of situations which can be recalled. A side effect of this view of the existence of a discovery process is that the mechanism must then be aware of the fact that it can look for a particular situation to use to form a needed node or to participate in a needed link. Noticing that a situation constrains another situation can come about via several modes:

1. Pragmatically by language use,
2. By direct observation,
3. By syntactic manipulation of situations known to the reasoner,
4. By creation of situations required by the reasoning process.

Now recalling that situations are themselves bindings over bindings this means that the matching process is one of examination of both property values and internal structures of the situation. It is the case that existential constraints between situations imply some sort of sharing of elements between the situations. In the simplest case the constraint between the two situations would require that every uniformity observable in the independent situation also be in the dependent situation. This implies then that matching over parts of the situations may be a mechanism for driving the reasoning process. This would make a nice explanation for an inductive reasoning process. Deduction on the other hand can be viewed as a powerful method for certain types of matching, particularly in its first order logic form. Such a view is consistent with the fact that even resolution refutation systems embody a technique (albeit systematic) for searching the fact space (ancestory filtered, set of support etc.) This would also make for an interesting explanation of why an individual does not believe everything that can be proven by deductive methods from his beliefs. There is simply no need to envoke the pattern matching process unless a particular situation type or link is being sought.

Since in general one thinks of reasoning as a method of extending the beliefs which one knows, a natural question is how does KAMSS "know" something. Simply put KAMSS can retrieve symbol strings from its memory. Given that those strings point to bindings or properties or stakes or higher level assemblages (e.g. situations, events, types etc.) KAMSS knows about the meaning of those strings as described in SRM. Thus in one sense the processing presented in section 5 for understanding of an utterance is one type of reasoning. When KAMSS "knows" something it may or may not know "how" it knows that something.

That is it may "know" that it has processed an utterance (or series of utterances) which resulted in the establishment of a binding with an associated set of properties or sub-bindings (as we saw in Section 5). Or it may know that the "thing" it has retrieved from memory is the result of a reasoned chain of information links between situations. This "optional" foundationism aspect of KAMSS allows the storage of justifications with a belief when there is some reason to doubt that belief.

In considering what triggers the reasoning process in KAMSS we assume that a reasoning process in KAMSS will always be triggered by an external stimulus. That is:

1. Trying to "understand" an utterance,
2. Trying to answer a question,
3. Trying to perform a task such as:
   - 3.1. Describing a system,
   - 3.2. Explaining a phenomena,
   - 3.3. Designing a model,
   - 3.4. Determining the consequences of a change.

Thus we are particularly interested in "goal-driven" reasoning. The "goal" itself is something that KAMSS knows about in more than a simple way. For example, KAMSS knows who was the source of that goal, and that it believes that it can satisfy that goal. Notice that the goals presented above are slightly different than one would normally encounter in a traditional reasoning system. In a traditional system the issue for the search algorithm is to find a path through a search space of beliefs to show that the current set of beliefs entail a goal sentence. In our case the beliefs represented in the knowledge base are links to entailments between real situations (among other things). That is, as was illustrated in Section 5, one of the pieces of information which the KAMSS derives from an utterance is that there exists a situation of the type described by the sentence in the utterance. Thus while we rely on the syntactic structures in the belief space for the method component (in particular the label strings) the discovery and the method component have access to the situation referents, which they can manipulate in useful ways. For example the method component can notice that two different labels reference the same situation or situation type (e.g. the fact that the "morning star" and the "evening star" both designate the planet Venus). On the other hand the discovery component can create a hypothesis that two identical labels which point to different situations or constraints actually point to the same situation or constraint type. Thus allowing the pursuit of evidence of the existence of that type in an inductive mode.

In considering the reasoning process for getting from what it knows to what it can reason to we must first make an assumption that if KAMSS makes an access to the information network described above then the result of that access is made a part of the network. We can think of this as the ability of KAMSS to talk to itself. That is, the access and the result of the access are made available for future accesses. Such a property is often referred to as "introspection" [Hass 1986; Moore 1980]. It is central to the "reasoning by change of view" (revision of belief) proposed by Harmon.

Having access to the interpretation as a situation of the type described by the use of a sentence allows the extension of the power of traditional proof methods. For example, *Substitution of Equals* can be applied at the situtation level rather than just at the syntactic or sentence level. We can apply propositional logic to the propositional statements associated with the interpretation independent of the variables which may be present in the situation type of a conditionalized interpretation.

In its simplest form reasoning by information flow then is a process of:

1. Knowing that one situation type S is informative on another S',
2. Knowing that a real situation s described by an abstract situation *s* of type S exists,
3. Concluding that a real situation s' described by an abstract situation s' of type S' exists.

Of course (as pointed out in the previous section) KAMSS has no way of getting at the real situation. Rather, KAMSS must deal exclusively with the situation descriptions. These descriptions are made up of sets of fact descriptions, each fact description being of a real fact supported by the real situation. In the previous section we noted that the abstract situations (the ones that KAMSS knows about because it has possession of their descriptions) come in three varieties: actual, factual and non-factual. By the definition of "actual" abstract situations if s and s' are actual abstract situations then they are identical since each is exhaustive. Thus, the interesting cases to consider are:

1. When s is actual and s' is factual,
2. When both s and s' are factual,
3. When s is factual and s' is actual.

In the first case the only additional information (derivable from knowing that s is informative on s') is that s' is (in fact) actual. In the last case knowing that s and knowing that S is informative on S' (with the background knowledge that s' is actual) corresponds most closely to what we normally think of as deductive reasoning as we can conclude the missing facts from s which are in s'. Case 2 with parameterization is actually the most interesting case. The reason is that in order for case 2 to support the sound deduction of additional information from knowing "that s" and knowing S is informative on S' we must establish what we refer to as the grounding constraints which transform s' into an factual abstract situation. It is the formulation of these grounding constraints which we feel correspond to the inductive reasoning process. The grounding constraints are just those constraints which make the relation (involves s, s';1) hold. Note that such a constraint does not force either s or s' to be actual. Rather it forms the conditions under which if s is realized so is s' and hence (since s' is factual) the facts supported by s' (the descriptions of which form the description of s') can be inferred. Grounding constraints are just in fact relations over real situation types. Thus, the description of a grounding constraint is just a collection of facts. This means that in the information flow style of reasoning, the grounding constraints can be manipulated directly. In other words, even the axioms and the proof rules can be objectified and reasoned about. To use computer science jargon, everything is a first class citizen. The importance of this feature has been documented in [Hass 1986; Barwise 1985b]. Hass, in fact, proposes an instance of this concept in his reflection schema proof method.

As discussed in previous sections generalized, quantifiers and conditional sentences are the kind of linguistic mechanisms used to establish grounding constraints. As such, these language constructs form relations between situations which account for the tranfer of information from one agent to another, or from one stage of the reasoning process to the next. With the relations described in the previous section the rules of inference for propositional logic can easily be formulated as constraints between situation type descriptions operating only on the proposition component of the description. Universal specification, modus ponens and other rules of inference can be added by adding the existential and universal quantifier relations.

6.5 Summary

In this section we have argued for the use of information transfer between situations as the basis for a method of reasoning. We have illustrated this mechanism as a means to support the types of reasoning required in KAMSS. The major features of this approach is that it allows for use of both the semantic and pragmatic interpretations of a statement to be used in the reasoning process as well as the fact that it supports objectification over the axioms and proof rules.

7. THEORY OF MODELING

The purpose of this section is to investigate the relevance of the previous sections covering situation theory and reasoning to the analysis of how intelligent beings perform modeling. If we assume the correctness of the basic assumption of attunement to uniformities across space and time as the basis for an organism to move successfully through its extent, then one of the challenges of this section is to distinguish the level of modeling which we are choosing to explain. In fact, the ability to be attuned to such uniformities can individually be characterized as a modeling task. It follows that the basic skill of language design and utterance formation can be conceived as a modeling task. Since we are interested in imitation of the process by which humans learn to perform these tasks or acquire this communicative competence, we will focus on this area first. We will then move on to a discussion of the generation of models for the causal reasoning and deductive simulation capabilities of KAMSS. Finally, we will summarize what these findings can tell us about the more traditional forms of quantitative simulation modeling.

7.1 Modeling and Semantics

Model-theoretic semantics involves the construction of abstract mathematical models of those things in the world making up the semantic values of expressions in the object language. Model theory is a method for learning about the meaning of expressions and the correlations between "expressions and meanings" by investigating in detail how the meaning of complex expressions is related to the meaning of the simpler expressions they are constructed from.

On the other hand, if we are interested in how models are used as generalized tools to help in decision making, we would focus on examination of what the previous sections have to say about how a person:

1. Formulates abstractions, associations, and model specifications,
2. Uses models for imitations, approximations, and indications of reality, and
3. Uses the imitations, approximations, and indications, in the reasoning process associated with his decision making.

It is this focus that the following section is intended to serve. However it should be pointed out that the two focuses are closely interrelated. The basic premise of this section is that humans use models in common sense reasoning about systems as a natural extension of the human language and reasoning capabilities. In fact the models provide a requisite tool for enabling common sense reasoning about complex systems.

In Section 5, we reviewed some of the characteristics of natural language from the perspective of Barwise and Perry [1983]. It is informative to note the similarities between models and languages relative to these characteristics as follows:

1. **Models as Situation Understanding Mechanisms:** The contention here is that models are used to key our perception mechanisms to be attuned to certain uniformities. One of the painful experiences that motivated this research was that sophisticated modeling support tools were not generally usable by domain experts because they did not have the requisite modeling experience. While it was feasible to train them in the basic concept classes of a modeling technique and in the structural aspects of a model, this training did not provide them with the ability to observe an actual situation and determine the design of a model having instances of those concepts so arranged as to answer a particular question. This is why a modeler can do what a non-modeler cannot. This captures the notion of "frames" as originally proposed by Minsky [1972].

2. **Parameterized Situation Descriptions as Model Templates:** Our attempts to derive information from a situation which has been perceived forces us into the process of parameterization of situations and postulation of grounding conditions which take the form of relations between parameterized situations [Barwise 85a]. The parameterization process leads to the concept classes within a modeling technique. The background condition formulation gives rise to the structural elements of a modeling technique.

7.1.1 The Role of Context, Viewpoint, and Purpose

One of the first things taught to modelers (and possibly one of the last things understood) is the importance of specification of the *purpose, viewpoint,* and *context* of a model. The purpose is usually described as an expression of the use for the model. The viewpoint characterizes the audience of the model. The context is the characterization of the boundaries of the model. In actuality the importance of these concepts is generally only recognized in the construction of the model where it is discovered that they are useful constraints for decision making. As a first level approximation, the context for a model can be viewed as a collection of states of affairs within which the facts represented in the model as assertions about the real world can be ascertained to be true. The concept of a context also can be viewed as an attempt to characterize the conditional constraints which define the background constraints under which the statements made in the model will be true. In contrast the viewpoint of a model is a characterization of the attitudes that the modeler attempts to reflect in a model. The "viewpoint" of a model is an encapsulation of the beliefs, knowledge, and terminology of a particular set of agents. Those agents are the representative set of individuals who could "understand" the elements of the model with respect to their perception of the "real world". The viewpoint of a model characterizes the set of individuals who would recognize the uniformities represented in a model as being consistent with those uniformities that are relevant in a particular slice of the real world. The "purpose" of a model is a characterization of the reasoning process which will be supported by information which can be acquired through the construction of the model.

7.2 Modeling for Causal Reasoning and Deductive Simulation

One of the basic tenents of this research was that the common sense notion of "simulation" was that of causal reasoning and deduction from an understanding of the structure and behavior of a system. The basic idea behind qualitative analysis, either using causal reasoning or qualitative simulation, is to be able to support as much analysis as possible on the phenomenologically naive description of the system. We do not propose that this qualitative analysis will completely replace a model based analysis. However, we do believe that:

1. In many cases the qualitative analysis will be sufficient or,

2. The qualitative analysis will indicate that the expense of the quantitative modeling is unnecessary or,

3. The results of the qualitative analysis will focus or guide the quantitative modeling or,

4. The qualitative analysis will be used as a mechanism for validation of the quantitative model based analysis.

We propose that this process can be imitated by the recognition that there are actually three different views of the world with which we are dealing. The first view is the "Description View" which is largely "type" and axiomatic in nature. In this view we capture what we know about the way things are and how they work. The SDC portion of the KAMSS is structured to acquire this view. The extensions to situation theory presented in Section 5 were designed to provide the basis for a representation system capable of storing such descriptions. The second view is referred to as the "Token View". This view captures models which we instantiate from the description view. Finally, there is a third view called the "Observation View". This level is structured to store assertions made about the structure or behavior of the system based on the results of tests or controlled observations. The role of each of these views will be described in the remainder of this section.

The "Description View" of the world (presented in Section 5) is adequate in structure and form to serve as the basis for deductive reasoning in a theorem proving fashion. However, many of the kinds of questions which the KAMSS must address either require information which will not be in the current state of the description base or they require possible world types of reasoning. To provide the type of reasoning necessary to answer questions which require design the KAMSS must support qualitative reasoning about either changes specified in the question or changes implied by a design process attempting to answer the question. For this, we need the ability to tokenize the type descriptions. Tokenization involves the instantiation of model objects for each type description and the treatment of the properties and relations of those objects as constraints which must be consistently applied across that tokenization. It is this token generation and manipulation process which we will refer to as "qualitative simulation."

Thus, for example, we would take a description of an event type such as:

truck cab arrival := at $\dot{l}$ during $\dot{t}$

cab, $\dot{a}$; yes

cab-carrier, $\dot{\mathbf{b}}$; yes

conveyor, body in white conveyor; yes

on, $\dot{\mathbf{a}}$, $\dot{\mathbf{b}}$; yes

on, $\dot{\mathbf{b}}$, body in white conveyor; yes

position, $\dot{\mathbf{b}}$; changing

as a schema from which we would generate an event occurrence (truck-cab-arrival-1), an object token (truck-cab-1), an object token (cab-carrier-1), and a constant (body-in-white-conveyor). The objects would be standing in the relations (on truck-cab-1 cab-carrier-1), and (on cab-carrier-1 body-in-white-conveyor). The event occurence would be assigned an interval, and because the description does not contain a specification of the duration, a default value of "unknown" would be assigned as the value of the duration.

The process concept introduced in Section 5 allows us to wrap what is traditionally referred to as *pre-conditions* and *post-conditions* around the state of change by describing a process type. For example, the process **truck cab arriving** consisting of the ordered sequence of a state of change and a state of affairs ⟨ **truck cab arrival, truck cab in position** ⟩, where the state of affairs would be described by:

truck cab arrival := at $l_{e}levator'$ during $\dot{t}$

cab, $\dot{a}$; yes

cab-carrier, $\dot{\mathbf{b}}$; yes

conveyor, body in white conveyor; yes

on, $\dot{\mathbf{a}}$, $\dot{\mathbf{b}}$; yes

on, $\dot{\mathbf{b}}$, body in white conveyor; yes

7.2.1 Generation of Qualitative Models from System Descriptions

Token based qualitative simulation requires the following elements:

1. A description of the completion criteria,
2. A description of the starting situation(s),
3. An algorithm for type instantiation,
4. An algorithm for constraint propagation,
5. An algorithm for the determination of whether the simulation process has reached the completion criteria,
6. A method for interpretation of the results of the process.

The automatic generation of qualitative models from system descriptions has many inherent problems. One of these problems is the characterization of the completion criteria of the qualitative analysis process. The description of the completion criteria is not necessarily straight forward. The completion criteria may exist in any one of the following forms:

1. A possible situation description (e.g., can this system process 80 jobs per hour),
2. An attribute value of a described situation (e.g., what is the throughput of the west end loader),
3. The discovery of a conditional constraint (e.g., under what conditions will the west end loader be a bottleneck),
4. The description of an operation of a system (e.g., what will happen if the BIW conveyor breaks down for three hours).

In Section 2 we delineated several cognitive tasks supported by application of internalized theories of system dynamics. We can see that the above completion criteria map to these tasks directly. Item #1 above is an example of *consistency analysis*, item #2 above is an example of *plan projection*, item #3 is an example of *situation anticipation* and item #4 is an example of *experiment planning*.

The need to have an initial state is at first examination a difficult need to fulfill. However, the initial state description can be satisfied by specification of the initial state of affairs of the system. The type instantiation process can be used as an effective mechanism for query of the user for this information.

Type instantiation is complicated by the global constraints which interrelate the states of affairs and states of change of the system. Space and time constraints being two obvious examples of such global constraints. The general process of type instantiation can be described by the following method:

```
Until no more instances can be generated or until the completion criteria is met
do:
 For each process type in the system description do:
  For each state of affairs or state of change in the process type do:
   For each object type in the state of affairs or change do:
   Create an instance of the object type,
   Validate the constraints associated with that object type,
  Create an instance of the state of affairs or state of change,
  Validate the constraints associated with that situation type,
 Create an instance of the process type,
 Validate the constraints associated with that process type,
Check the completion criteria,
Continue.
```

Of course in practice the simplicity of this method is complicated by the specification of the actions which must be taken on the discovery of a constraint violation. The default action is to invalidate the associated instantiation process (lazy evaluation); alternatively the products of the instatiantion process can be set aside and an attempt made later in the process to reuse these tokens.

7.2.2 Example of Qualitative Simulation Process

In this section we provide a walk through of the above described process for a sample situation. Suppose we are faced with the following problem. We have a transfer system which transfers truck cabs from the Body-in-White conveyor to the Uniprime conveyor. We want to know if this mechanism will transfer the cabs at a particular rate (say 60 cabs per hour). One obvious way to answer this question is to look up the designed transfer rate of the system in question. Unfortunately this will often fail to produce the information we want because what we perceive as a "transfer *system*" may not have been specified, designed, or certified as such. It is quite likely that the only information that is available relative to capabilities is information at the individual machine or mechanism level. In which case our next step is to walk to the shop floor and notice the following actions occurring:

1. Cabs arrive at the transfer location on cab carriers on the BIW conveyor,
2. The forks of the transfer mechanism extend beneath the cab,
3. The cab is lifted off of the cab carrier by the forks,
4. The forks with the cab on them retract to a home position,
5. The elevator raises the cab to the transfer level above the Uniprime conveyor,
6. The forks extend out over the Uniprime conveyor,
7. The cab is lowered onto a carrier on the Uniprime conveyor,
8. Cabs depart from the transfer point on cab carriers on the Uniprime conveyor.

It is important to note that the above action descriptors describe action types (i.e., their descriptions would contain parameterized situations). One of the first principles of qualitative modeling is to account for all objects. In the above description of activities associated with the transfer process, the following actions must be added to account for the carriers:

1. The empty cab carrier leaves the transfer location,

2. Empty cab carriers arrive at the loading location on the Uniprime conveyor.

If one begins to build descriptions of the transfer process, its associated actions, and the objects in the system, one would end up with something like the following. We will assume that the general schema of a description follows that outlined in Section 5. Thus a description of a concept will contain the following constituents:

1. A list of facts,

2. A set of axioms for membership,

3. A set of axioms of membership.

In the example, the axioms for membership will be used as "axioms for instantiation" (i.e., axioms which guide the determination of when a tokenization of a concept type can occur). The "axioms of membership" will be used to determine the viability of an instantiation once it has been made. Since we are to operate in a "token" world we will find that some of the "axioms for membership" will specify what the conditions are for destruction of an instance, and likewise, part of the "axioms of membership" will describe what happens when the instance is destroyed. An alternative approach would be to place these axioms into the definition of a new concept type associated with the original concept type which represented the termination instances. However, we have chosen the use of termination axioms in our experimentation. In the ensuing example we will introduce several relations (e.g., same, common, etc.) which would have to be formally defined in an actual KAMSS system.

First, one can list the actions within the process:

1. Cabs arrive at the transfer location on cab carriers on the BIW conveyor,

2. The forks of the transfer mechanism extend beneath the cab,

3. The cab is lifted off of the cab carrier by the forks,

4. The forks with the cab on them retract to a home position,

5. The elevator raises the cab to the transfer level above the Uniprime conveyor,

6. The empty cab carrier leaves the transfer location,
7. empty cab carriers arrive at the loading location on the Uniprime conveyor,
8. The forks extend out over the Uniprime conveyor,
9. The cab is lowered onto a carrier on the Uniprime conveyor,
10. Cabs depart from the transfer point on cab carriers on the Uniprime conveyor.

Thus, we will use act1 as the name of the activity type of "cabs arrive at the transfer location on cab carriers on the BIW conveyor". We will use the synonym "pTC" to refer to the "transfer cab" process type. We will use [] to denote a variable representing "an" (actually "any" one) instantiation of a parameterized description. We need a predicate to allow us to refer to something which is constant over all instantiations. We also need a notation to allow us to refer to "a" specific instantiation. We will prepend a # and append a number to refer to a specific instantiation. Thus [# cab-1] refers to the cab (token of course) labeled cab-1. Since descriptions are collections of things, it is useful to be able to refer to components of a description. We will use a path notation for making such a reference. Thus,

pTC.act1

refers to the cab arrival activity description within Transfer Cab description. Similarly:

[# pTC-1][# act1-1].[# cab-1]

refers to the cab (labeled cab-1) in the cab arrival activity of the first instance of the transfer cab process.

The description of the overall process would be:

```
(Process Transfer Cab
        (constituent-sequence pTC, (act1,act2,act3,act4,act5,
                                    act6,act7,act8,act9,act10)
                                    ; 1)
        (common [cab], [pTC];1) -- meaning the instance of the
                                   cab type is the same instance
                                   over all action instances in
                                   the instance of the process
                                   pTC.
        (common [BIW-cab-carrier], [pTC];1)
        (common [Uniprime-cab-carrier], [pTC];1)
        (common [BIW conveyor], pTC;1) -- meaning the instance
                                          of the BIW conveyor
                                          is the same over all
                                          instantiations of pTC.
        (common [Uniprime conveyor], pTC;1)
        (common [forks], pTC;1)
        (common [BIW transfer location], pTC;1)
        (common [Uniprime transfer location], pTC;1)
        (before [pTC].[act7] [pTC].[act8];1) -- meaning an
                                                instance of the
                                                arrival of a
                                                Uniprime cab
                                                carrier need
                                                only precede
                                                the forks
                                                extending over
                                                the uniprime
                                                conveyor
        (after [pTC].[act6] [pTC].[act5];1) -- meaning the
                                               instance of the
                                               departure of
                                               the BIW cab
                                               carrier need
                                               only follow the
                                               raising of the
                                               cab, thus all
                                               of the follow-
                                               ing actions
                                               can take place
                                               in parallel to
                                               act6.
        )
```

Each of the activity descriptions would take the following form.

```
(Activity cab arrival
        (at-location act1, [BIW transfer location];1)
        (agent act1, [BIW conveyor];1) -- the body in white
                                          conveyor
        (object-1 cab) -- something of type cab
        (object-2 BIW-cab-carrier) -- something of type cab
                                      carrier
        (on [act1].[cab], [act1].[BIW-cab-carrier];1)
                                -- the instance of a cab involved
                                   in the arrival activity
                                   is on the BIW cab carrier
                                   involved in that activity
        (at-time arrival-time;1)
        (duration arrival-time, Rsec;1)
                                -- the duration of this activity
                                   is below the resolution of a
                                   seconds based measurement
                                   system
        )

(Activity forks extend
        (at-loc act2, [BIW transfer location];1)
        (at-time extend-time)
        (object-1 cab)
        (object-2 [forks])
        (duration extend-time, [act2], 15 seconds;1)
        )
```

Note that there are several ways in which constraints between realizations of the types in the above descriptions can be formulated. For example, given that one process is ongoing, how is the generation of another transfer process instance constrained. In the collection of system descriptions, it was noted that the techniques used were either explicit or implicit. A statement such as: *The empty BIW cab carrier must clear the loader area before another transfer process can begin* represents an explicit constraint. Explicit constraints are the easiest for the reasoner to handle since they can be directly used as input as additional top level process instantiation constraints. Often background conditions, referred to as "domain constraints," were exploited in the implicit cases (e.g., the fact that only one object can occupy the BIW transfer location at a time). In the above example, an instantiation of the transfer cab process must involve the instantiation of a cab arrival event which occurs at a specific location.

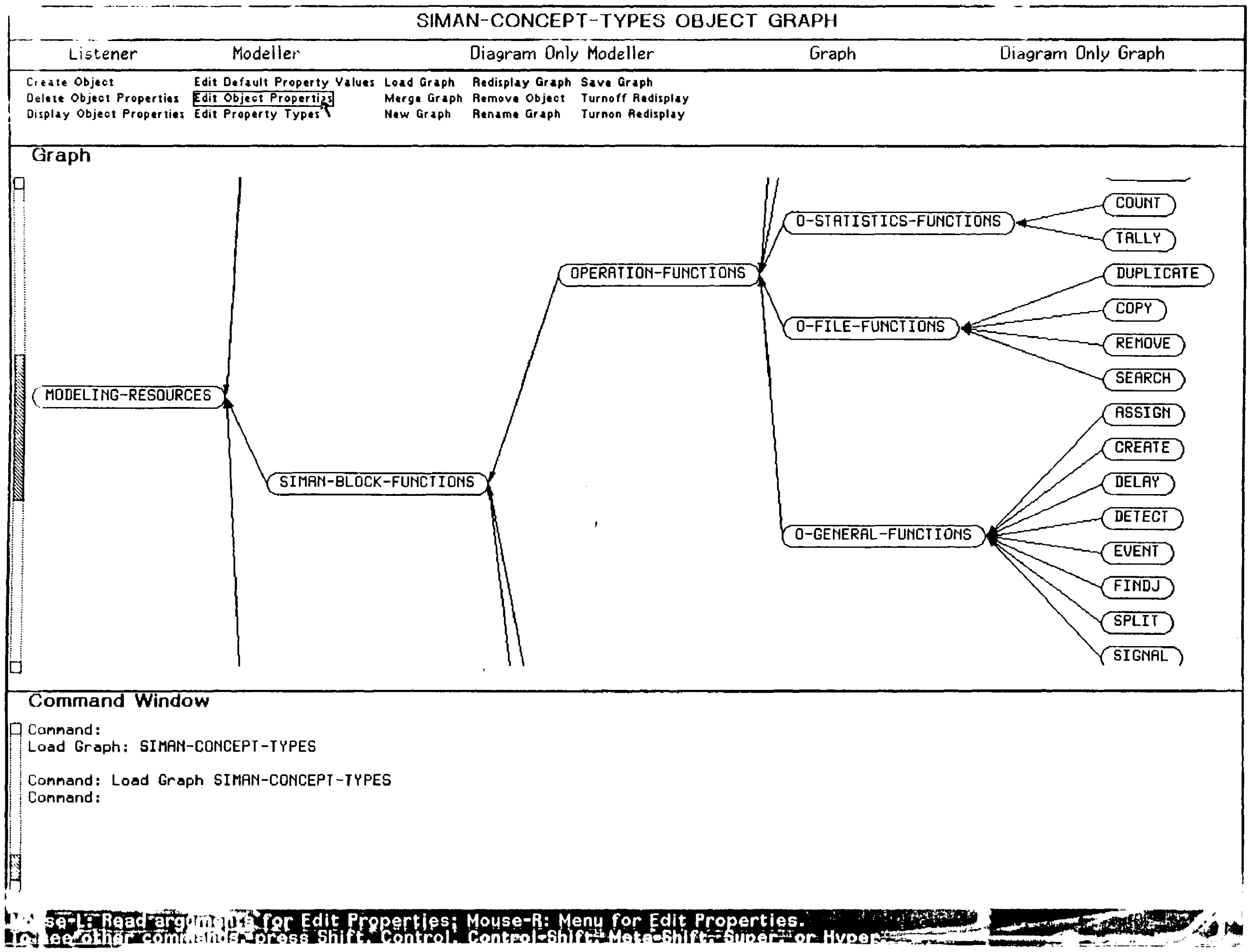
The qualitative simulation proceeds by first making an instance of all of the constant concepts in the pTC description. It then makes an instance of pTC.

It proceeds for each activity in the activity sequence to make an instance of the activity, generating all of the necessary object instances, and checking the constraints as it does so. For example on generating [act1], it must generate [cab]. As it does, it checks the constraints associated with the process instance [pTC] and finds no violations. However, in the attempt to generate an instance of [act2], it would be required to generate another instance of [cab] which would violate the "common" restriction of pTC and hence cause a failure. On the other hand, in attempting to generate an instance of [act7], it would find no such violations and would henceforth generate such an instance and the associated object instances.

7.3 Quantitative Simulation Model Design

A discussion of model design cannot be attempted without taking into account such practical issues as model type, purpose, resource availability, and intended use. For the purpose of this section, we will consider entity flow mathematical simulation models or the type of model most often used in queueing network studies. Our contention is that the model type provides a template or set of scripts which form a classification scheme. The task of the model designer is to fit the system description into the structures provided by that classification structure. That is, he must map the situations observed or known about into the abstract situation structures provided by the modeling scheme. The more extensive the classification scheme provided by the model type, the more closely the modeler can fit the real world structures into that scheme. From this point of view we can see how the modeling structure both supports the abstraction process and also drives the perception process of the analyst. To see how this might work, consider the IDEF1/ES model of a subset of the SIMAN^{TM} language [Pegden 1982] displayed in Figure 7.1. Given that the simulation expert has acquired the knowledge of the concepts and stuctures presented in that figure, we can discuss how those constructs are used in the mapping process to result in a mapping which gives rise to the "level of abstraction" property often ascribed to a model. The following discussions characterize the selection of the "level of abstraction" as a series of abstraction mechanisms which the modeler uses during the model formulation process.

Structural abstractions: In Figure 7.1, we see that the central component of the SIMAN model is an activity concept. If there were no stated purpose



```
Command:
Load Graph: SIMAN-CONCEPT-TYPES

Command: Load Graph SIMAN-CONCEPT-TYPES
Command:
```

```
se-L: Read arguments for Edit Properties; Mouse-R: Menu for Edit Properties.
To see other commands, press Shift, Control, Control-Shift, Meta-Shift, Super, or Hyper
```

FIGURE 7.1: CONCEPT MODEL OF SIMAN.

or usage constraints on a model designer, he might attempt to map the entire system into the activity concept structure. Given that he made such a decision the model structure imposes constraints on the additional information which must be provided to satisfy the model completion criteria. Trying to satisfy these information constraints would require him to make additional observations on the system. These observations force the recording of bindings which are not necessarily part of the normal system perception process. Given that the modeler can make these observations and that there are resources available to make them, there is no problem. The more likely case is that it is impossible or expensive to make the observations directly (e.g., record the number of strikes over the next 20 years). In this case, the modeler must then decide how to approximate the input required. This is where experience comes in. For example, an experienced modeler knows that a good starting point for modeling a distribution is as a triangular distribution. Such knowledge may never be taught in the classroom because, as with most expertise, it is fraught with theoretical problems. However, the experienced modeler knows that the problems occur so seldomly that making the assumption, particularly as a part of an initial model, is reasonably justified. This is also where the simulation model specification language comes to the assistance of the modeler. The language provides guidance relative to the types of structures that are required (e.g., an activity must receive an entity from some other model component) and provides options for satisfaction of these constraints.

Attribute Selection: In the case where the desired information output is defined by the problem statement, the intended use, or the purpose of the model, the task of the modeler is to choose a set of modeling components which will produce that information. Thus, for example, if we are interested in the average number of parts in a machine buffer, we know that the most direct way to calculate this statistic is to represent the parts directly as model entities. However once this representation is selected, there is the decision relative to what attributes of the part to carry over to the model entity. In the internal representation of the description of a part, there are any number of properties associated with a part (the average number of tracked attributes in a manufacturing information system is over 30). From the model structure, however, we know that for this particular statistic only one attribute is actually needed–the unique identifier of the entity. Most often the rules of the modeling language provide for the automatic generation of a unique identifier for each entity created as illustrated in Figure 7.1. Even this attribute is ignored from the system description.

Boundary setting: Another type of abstraction mechanism which the modeler uses is related to the determination of the boundaries of the system. Wherever the modeler draws the boundary from the modeling point of view, he must account for the rest of the world by the interface specification across that boundary. The process of boundary setting is somewhat simpler in human discourse situations. In the communication process the speaker need only communicate (describe) those situation(s) which entail the situation being described back to a point where the listener shares common knowledge with the speaker (i.e., back to a common resource situation). We can characterize the boundary setting activities of the modeler by analogy to this communication process. Given any model situation, such as an assignment of the milling center to an activity in the concept model displayed in Figure 7.1, the model structure dictates that the activity concept must receive entities from somewhere. The question which the modeler must answer is whether or not the "situation(s)" relevant to the production of those entities provide any interesting constraints on the modeled situation or entailments other than the furnishing of the entities. If they do, the modeler will generally represent them explicitly and hence push the boundary of the model further out. Interesting constraints include those where the modeler knows from the system description that the two situations share common components which are often described as a resource contention or where specific conditional constraints exist.

Granularity of ordering mechanisms: Another type of abstraction which the modeler must deal with is the granularity of the "stakes" in the model. Most obviously this includes the time and space stakes, but it also includes the individualization of the parts which is chosen. That is, for example, how finely the part is broken down into the subcomponents of that part. The modeler may choose to represent the movement of engines through an assembly area as single items without representing the components which come together at each assembly point to create a more complete engine.

Substitution of constraints: "Substitution of constraints" is an abstraction mechanism which the modeler can and generally must use. Substitution of constraints occurs when the modeler substitutes a constraint provided in the modeling language for constraints observed in the system description. A good example of this type of abstraction mechanism is the use of bounded counts as a mechanism for enforcement of physical spatial natural constraints (i.e., two physical things cannot be in the same place at the same time). Thus, the physical limit on the number of parts in a machine buffer is modeled as a counter with an upper bound.

In all of the abstraction mechanisms described above, there is one underlying mechanism which the modeler can and does employ. This is the mechanism of deciding that enough is enough and proceeding on with the test of his model by actually executing an analysis of the model (at whatever level of abstraction it exists) and comparing the results of that model against actual observations.

7.3.1 Summary

In this section we have presented a case for models as mechanisms for situation understanding and the use of parameterized situations as model templates. We proceeded to show how these model forming mechanisms could be used as the basis for qualitative reasoning. We outlined two approaches to qualitative reasoning about discrete processes from parameterized situation descriptions. Finally, we discussed the insight which this conceptualization of the modeling process provides for the areas of viewpoint, purpose, and abstraction setting in quantitative modeling.