AD-A219 785

CONCEPTUAL MODELING VIA LOGIC PROGRAMMING

Logicon

John Burge, Bill Noah, Les Smith

DTIC
S ELECTE
MAR 28 1990
B D

90   195

This report has been reviewed by the RADC Public Affairs Division (PA) and is releasable to the National Technical Information Services (NTIS) At NTIS it will be releasable to the general public, including foreign nations.
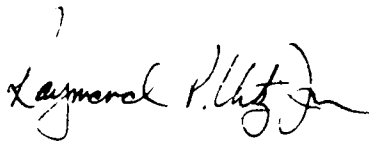
RADC-TR-90-11 has been reviewed and is approved for publication.
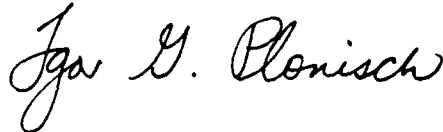
APPROVED:

JAMES L. SIDORAN
Project Engineer

APPROVED:

RAYMOND P. URTZ, JR.
Technical Director
Directorate of Command & Control

FOR THE COMMANDER:

IGOR G. PLONISCH
Directorate of Plans & Programs

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED			1b. RESTRICTIVE MARKINGS N/A		
2a. SECURITY CLASSIFICATION AUTHORITY N/A			3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.		
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A					
4. PERFORMING ORGANIZATION REPORT NUMBER(S) LSIS890166			5. MONITORING ORGANIZATION REPORT NUMBER(S) RADC-TR-90-11		
6a. NAME OF PERFORMING ORGANIZATION Logicon	6b. OFFICE SYMBOL (If applicable)		7a. NAME OF MONITORING ORGANIZATION Rome Air Development Center (COEE)		
6c. ADDRESS (City, State, and ZIP Code) 255 W 5th St San Pedro CA 90731			7b. ADDRESS (City, State, and ZIP Code) Griffiss AFB NY 13441-5700		
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Rome Air Development Center	8b. OFFICE SYMBOL (If applicable) COEE		9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F30602-87-D-0092 (CSC Task)		
8c. ADDRESS (City, State, and ZIP Code) Griffiss AFB NY 13441-5700			10. SOURCE OF FUNDING NUMBERS		

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO	WORK UNIT ACCESSION NO.
62702F	5581	QB	01

11. TITLE (Include Security Classification)

CONCEPTUAL MODELING VIA LOGIC PROGRAMMING

12. PERSONAL AUTHOR(S)
John Burge, Bill Noah, Les Smith

13a. TYPE OF REPORT Final	13b. TIME COVERED FROM Sep 87 TO Mar 89	14. DATE OF REPORT (Year, Month, Day) January 1990	15. PAGE COUNT 176

16. SUPPLEMENTARY NOTATION

N/A

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Conceptual Modeling
12	05		Logic Programming Prolog
25	05		Command and Control

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

The Conceptual Modeling via Logic Programming Task was to determine whether it is feasible to use conceptual models as a design aid for command and control systems, and whether these conceptual models can practically be implemented using logic programming. The major efforts included a review of conceptual modeling and logic programming, development of concepts for using conceptual modeling to aid command and control designers, development of a demonstration system, and evaluation. A Quintus Prolog-based demonstration system was built. The feasibility of using conceptual modeling implemented in logic programming was established.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ OTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED	
22a. NAME OF RESPONSIBLE INDIVIDUAL James L. Sidoran	22b. TELEPHONE (Include Area Code) (315) 330-2762	22c. OFFICE SYMBOL RADC (COEE)

DD Form 1473, JUN 86
Previous editions are obsolete.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

This document is the final report of a research task performed by Logicon for the COEE Branch of RADC's Command and Control Directorate. The research concentrated on investigating the feasibility of using Conceptual Modeling implemented via Logic Programming, or CMLP, as a design aid to improve U.S. capability to develop command and control systems.

1.1 PROBLEM STATEMENT

Weapon systems must evolve as the threat they were designed to counter evolves. Much of the change is implemented in the command and control elements to allow existing sensors and weapons to be employed in new ways. In like manner, changes in weapon and sensor systems almost always force a concomitant change in command and control.

Command and control system designers usually rely on simulations to evaluate alternative concepts and point designs. However, simulations are expensive and difficult to use, and often they operate at too low a level of detail to provide meaningful results to the C^2 system designer. The emerging technology of conceptual modeling may prove valuable by providing high-level models of important C^2 features. That is, conceptual modeling may aid the designer in forming a perception of the system, and thus may greatly benefit the design process.

1.2 APPROACH

The CMLP project consisted of three interrelated investigations:

- Collection of C^2 system design information and design processes
- Investigation of the capabilities of conceptual modeling and its implementation through logic programming
- Development of an experimental system to gain hands-on experience with the conceptual modeling of C^2 systems

The CMLP project was in practice divided into the following seven tasks, with the last further subdivided into six subtasks. Figure 1-1, the project schedule, indicates planned and actual task completions.

1. **Define C^3I Model.** This task collected a variety of data on the nature of C^3I systems. To bound the task, we concentrated on C^3I systems performing air defense and space defense missions, for each mission defining the hierarchy of the C^2 system, the nature of the threat, the functionality and implementation of the sensors and weapons interacting with the C^3I system, and the functionality of each element of command and control, communications, and, to the extent that information was available on an unclassified level, intelligence. We also defined generic performance goals for evaluating C^2 systems. With RADC concurrence, we further bounded the problem by not collecting cost data, and by excluding communications and intelligence from the remainder of the project. This narrowed the focus to a limited research area for better evaluation of the application of CMLP technology. The full set of data collected during this task was presented in progress reviews. We incorporated a substantial subset in the demonstration model after making the information generic with applicability across a variety of C^2 elements for defensive systems. Appendix A describes the information used in the model.

2. **Investigate Logic Programming.** This task surveyed conceptual modeling concepts and logic programming techniques. Section 2 describes the task outputs.

3. **Define User Interface and Query Language.** This task initially considered a line-by-line interface that would be portable from machine to machine. However, we decided that portability was less important than providing a flexible, easy-to-use interface that would encourage repeated use of the demonstration tool and also encourage future implementations. We therefore changed to ProWINDOWS, an interface using a powerful window structure. The interface was limited to run on workstations manufactured by Sun Microsystems, Inc. Appendix B, the User's Manual, describes the interface concept.

Task	1987			1988												1989			
	O	N	D	J	F	M	A	M	J	J	A	S	O	N	D	J	F	M	A
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1. Define C³I Model																			
2. Investigate Logic Programming																			
3. Define User Interface and Query Language																			
4. Define Procedures for Specifying Output																			
5. Select Logic Programming Language																			
6. Develop Methodology for C³I Users																			
7. Design and Develop Model																			
7.1 Develop Knowledge Base																			
7.2 Develop Query/Update																			
7.3 Extend the Database																			
7.4 Provide Data Dictionary Access																			
7.5 Provide Output Capability																			
7.6 Evaluate Model																			

(Chart annotations: "Line by Line" and "Windows" appear within the row for task 3.)

▭ Original Plan   ▬ Actual

89-02-121e

Figure 1-1. CMLP Project Schedule

4. Define Procedures for Specifying Output. This task concentrated on specification of C^2 data in a knowledge base that is easily accessible by the user. The outputs of a model with a user-friendly window-oriented interface are defined by the interface itself.

5. Select Logic Programming Language. This task performed a trade study of the available logic programming languages and selected the Quintus implementation of the Prolog language. Section 3 discusses the goals, model design, and concept for using the model.

6. Develop Methodology for C^3I Users. This task specified the nature of the user, developed a concept of user operation, and produced the user's manual presented in Appendix B. The task was updated following input from RADC on the nature of the conceptual models and the use of ProWINDOWS.

7. Design and Develop Model. This task, whose results are described in Section 4, had six subtasks, as follows. The task produced an Operational Concept Document, Software Requirement Specification, and Software Top Level Design Document.

7.1 Develop Knowledge Base. This subtask defined the design of the knowledge base for each of the models and for multiple sessions and baselines. It was changed considerably with the advent of the window interface and the ability to interact between elements in the knowledge base by the use of multiple windows.

7.2 Develop Query/Update. This task developed a query/update capability featuring

easy access to and change of all parameters within the knowledge base.

7.3 Extend the Database. All early development work on the knowledge base used the air defense mission. This subtask collected information to populate databases for space defense and for extended air defense in Europe.

7.4 Provide Data Dictionary Access. This task provided capability for data definition within the window forms, and extensive help capabilities.

7.5 Provide Output Capability. This task provided the capability to print the data in the knowledge base.

7.6 Evaluate Model. This task focused in priority order on evaluating (1) the extent to which the techniques involved in Conceptual Modeling via Logic Programming can be applied to a design problem such as command and control, (2) the extent to which CMLP can effectively model C^2 and other systems, and (3) the demonstration model as built. Section 5 describes the extent to which we used conceptual modeling and logic programming in the demonstration and evaluates the potential of conceptual modeling and logic programming for a C^2 design.

2. CONCEPTUAL MODELING AND LOGIC PROGRAMMING

This section provides an overview of conceptual models and modeling methodologies, both independent of and dependent on the problem domain being modeled; presents observations about the process of conceptual modeling drawn from our experience in modeling; presents two alternatives for constructing a model within a chosen domain; and considers logic programming and other techniques for implementing conceptual models.

2.1 DEFINITION AND PURPOSE OF CONCEPTUAL MODELS

Conceptual models allow us to create and manipulate an abstract representation of a hypothetical or real-world system at a level of abstraction comparable to a human expert's conceptualization of a problem-solving domain (Brodie, Mylopoulos, and Schmidt, 1984). A conceptual model consists of a number of symbol structures and symbol structure manipulators corresponding to entities, attributes, and relationships of the world of a human observer (Borgida, Mylopoulos, and Wong, 1984). We distinguish between conceptual models and other uses of the term "model," such as the traditional mathematical model (a set of mathematical entities and relationships that satisfy a given set of axioms) or the analog model (e.g., a wind-tunnel model).

Conceptual modeling is conducted independently of the final implementation of the model in software, in hardware, or on paper. (We will restrict subsequent discussion to the first of these representation possibilities.) In this respect it is similar to the use of semantic data models that permit the specification of database elements and relations before an actual design specification has been achieved. It is also similar to the specification of computer programs in terms of basic constructs (data types, control flow, variables, functions, etc.) before implementing the

constructs in a particular language, or even before choosing the language.

The impetus for conceptual modeling has arisen from the increasing complexity and precision that can be supported by artificial intelligence—specifically, the ability to represent the knowledge of a human expert in a problem-solving domain. This knowledge is no longer confined to facts and problem-solving algorithms, but includes goals and subgoals (embracing strategies and tactics), heuristic knowledge, and meta-knowledge. Exploiting these capabilities requires communication between domain experts and knowledge engineers, and between knowledge engineers and end-users of AI-based systems.

Conceptual modeling is the basis for such communication. Expert system shells and AI development environments point the way toward automated support for conceptual modeling, and thus toward broadening the communications channel between the key players in the expert system development cycle. This widening is essential to resolving the major bottleneck in the development of intelligent systems and expert planning and decision aids: getting knowledge out of the head of the human expert and into the system.

The following discussion, based on the combined experience of many researchers in the field, including the CMLP team, provides indications of the nature of conceptual modeling and current limitations on its usefulness. It also indicates the technical requirements that remain to be met if conceptual modeling is to become a well-defined tool for system development, much as top-down methodologies for analysis, design, and implementation have become (Yourdon, 1979). We begin an increasingly detailed examination of these issues by looking at the meta-level of modeling, that is, at the aspects of conceptual modeling that are common across application domains.

2.2 THE NATURE OF CONCEPTUAL MODELS: DOMAIN-INDEPENDENT ASPECTS

While we can speak in general terms about conceptual modeling from a domain-independent perspective, we must nevertheless acknowledge that, like much of human activity, constructing conceptual models is an inherently goal-driven process. This is no idle philosophical point: specific decisions concerning basic epistemological issues, the selection of a theory of modeling, the methodology for model creation, the selection of data used to create the model, the interpretation of these data, and the structures used to implement the interpretation are all influenced by the problem-solving environment within which the model will be put to use. To fully understand conceptual models, therefore, we can benefit from examining a set of "first principles" that put conceptual modeling in a broader context before going on to issues of knowledge representation and methodologies.

Gaines (1988) describes humanity as a "distributed anticipatory system," that is, a coordinated set of semiautonomous systems whose primary social drive is the anticipation of the future. This arises from basic survival needs; the human community must anticipate the future if it is to react to or alter it (the latter capability perhaps being a principal criterion of "human intelligence" that could be applied to "machine intelligence"). A strategy of information control has evolved which optimizes the comparison of threats and resources; its goal is to minimize uncertainty at the levels at which reality can be modeled. This characterization provides the systematic foundation of general theories of modeling.

It is important to note that this approach emphasizes the social nature of modeling. All conceptual models are, by definition, based on human conceptualizations of reality. All models are therefore to some degree subjective, even at the most primitive level of physical measurement (e.g., as subject to the limitations of the uncertainty theory and the propositions of quantum mechanics). We believe that even the direct modeling of a physical system (e.g., inferring the structure of a stochastic automaton from its behavior) is qualified by cultural, organizational, personal, and cognitive biases that ultimately derive from the foundation delineated above. Therefore, it is impossible to speak of conceptual modeling independent of the socially derived personal constructs of the human source of the knowledge that resides in the model. Typically, this source is the domain expert whose knowledge is first elicited, then reorganized and expressed, by a knowledge engineer.

The role of the domain expert in the anticipatory social dynamic that we have described is a special one. Knowledge in the system increases on the basis of induction; only induction creates new knowledge and therefore reduces uncertainty. We may paraphrase Goodman's (1973) notion of reflective equilibrium (as interpreted by Gaines) as: "Knowledge is amended if it produces an inference with which we cannot agree; an inference is rejected if it leads us to reject knowledge without which we cannot live." These judgments cannot, however, be supported on an individual basis—the result would be chaos in the form of a truncated shared conceptual model among the members of society. To accommodate this problem, the community of domain experts serves as the point of reference for the evolution of our socially shared knowledge base. They are the arbiters of the knowledge acquisition process.

To provide guidelines for using conceptual models and assessing the scope of the problem space they can address, we need to understand this process. Gaines has provided one theory of modeling which integrates epistemological, methodological, action, cultural transmission, and knowledge transmission hierarchies. This theory is well suited to the social nature of conceptual modeling and provides a framework on which we can build a detailed study of conceptual modeling from both

domain-independent and domain-dependent perspectives.

Gaines's view of modeling is based on the interplay of *events* (the flow of uncertainty up the hierarchy) and *actions* (responses to uncertainty flowing down). This polarity can be seen as representing bottom-up *informing* (and its antithesis, *surprising*) and top-down *structuring*, corresponding to efferent and afferent neural channels. Yet, we must keep in mind that at each level these channels interact. Perception (of an event) is as much a structuring of the sensorium as it is the passive transmission upward of signals, a fact which is true socially as well as individually. The agreed-upon structure of knowledge (i.e., the socially shared conceptual model), arbited by the community of domain experts, influences what is seen, and therefore what is modeled and how the underlying reality is acted upon through the model.

We can summarize Gaines's approach by combining the different hierarchies at each of six levels.

Epistemologically, the first level is the *source system*, based on the distinctions among the fundamental concepts (or personal constructs) that make it up. These distinctions are made in the course of actual interactions with the world. Cognitive skills at this level are basically reflexive, based on stimulus-response pairs and associations between constructs and primitive acts. Knowledge is transmitted culturally through the informal mechanism of mimicry and behavioral modeling. Experts transmit their knowledge similarly, by example.

The second level is the *data system*, based on distinctions made about an event in the source system. The level can be seen as the basic representation of objects and relations. These distinctions are made on the basis of our experiencing of (including measuring) the source system. Cognitive skills at this level involve linking constellations of experience into complete action sequences on the basis of generalization of similar experience. Knowledge is transmitted culturally by reinforce-ment, rules, or induction based on events in the source model. The expert transmits knowledge by working with and supervising an apprentice.

The third level is the *generative system*, which yields productions that predict events in the source system in terms of the data system. The match between these predictions and the events propagating up the hierarchy signifies the accuracy of the model. This level represents hypotheses based on experience. The cognitive skill at this level is the ability to build an optimal action sequence. Culturally, knowledge is transmitted by technical means (i.e., rational explanation); the expert transmits knowledge through the literature and responses to questioning.

The fourth level is the *structure system:* the comparison of generative models, for instance in terms of simplicity. This level is achieved through correspondences (or *analogies*) among hypothetical rationalizations. The cognitive skill involved is goal-based *comparison* of alternative models. Culturally, knowledge is transmitted by simile and metaphor—by *transfer* from related domains. The expert transmits knowledge by using analogous models drawn from similar problem-solving situations.

The fifth level is the *meta-system*, which contributes the basis for comparing models. It is acquired through *abstraction* (which is the foundation of analogy). The cognitive skill associated with this level is the creation of *abstract models* in the form of templates which can be instantiated to satisfy a given goal set. Knowledge is transmitted culturally through *formalisms*, such as mathematical laws. The expert also transmits knowledge in the form of formal laws.

The sixth level (and potentially higher levels) is represented by the *meta-meta system*, which specifies further relations on the levels below. For instance, such a system might provide contrasting theories of knowledge for comparing models with different criteria. This level is achieved through *transcendencies*, or accounts of abstractions.

The associated cognitive skill is the recognition of the entire process as being level dependent, and the ability to generate new relations among levels; this requires being able to stand outside the framework (of the lower levels). Knowledge is culturally transmitted by system-theoretic analysis or by transcendent experience. The expert transmits knowledge through the process of deriving formal laws.

The principal uses of this modeling theory are to emphasize the hierarchical nature of conceptual modeling and to map this nature onto the technical means for constructing a model. Gaines suggests a hierarchy of knowledge engineering techniques already in use that correspond to the six levels:

1. *Knowledge base:* facts and inferencing rules

2. *Expert system shell:* operational system for applying the knowledge base

3. *Inference system:* derives the consequences of the facts

4. *Planning system:* determines how to use the inference system to satisfy the goals

5. *Explanation system:* answers questions on the basis of the inferences made

6. *Knowledge acquisition system:* the processes used to establish the knowledge base (and establish the inference and planning strategies)

Gaines's approach results in some foundations for knowledge engineering that are worth adapting from knowledge acquisition to conceptual modeling (which are very similar activities):

• There is a detailed structure to the cognitive processes used in conceptual modeling. There are links between these processes and neural processes in the brain.

• The basic system that should be considered in modeling includes a social component in terms of which the modeling effort will inevitably be organized. This component must therefore be recognized.

• Cultural and expert knowledge transmission proceed in explicitly identified ways during model construction.

• Groups of exper   rather than individuals, should be used in developing conceptual models to capture social, cultural, and experiential variance.

• Methodologies currently available can be incorporated into conceptual modeling efforts, analyzed, and compared.

The hierarchical view of conceptual modeling brings up another important issue: as the philosopher Korzybski pointed out, there is a distinction between the map and the territory. Or, as Bateson (1979) noted, all modeling is a coding, or transformation, between the model and the thing being modeled, the *Ding an sich*. This coding involves separating a system into its components: recognizing parts and wholes. Such activity, Bateson showed, is a convenience—there is nothing intrinsic in nature which necessitates any particular dissection. We therefore suggest that one must look to the context of a particular model to discern the reason for a particular choice of objects, relations, facts, rules, inference mechanism; etc.

What must be taken into account is that the matrix provided by the choice of representational media and the decisions of the components to be discretely represented inevitably distorts the system being modeled. Consider, for example, a mapping exercise in which land masses are mapped onto a coordinate system. The resulting model will appear quite different if the coordinate system resides on a flat matrix, as opposed to a toroidal one. Bateson points out that any matrix, including a language or propositional system, will in principle distort the system being mapped onto it. This is the Procrustean bed of conceptual modeling: the thing being modeled is shortened or lengthened to fit the model in which we strive to put it.

With this view of conceptual modeling as background, let us discuss some of the more concrete aspects of conceptual models. We will examine these aspects from the perspective of the top levels of our hierarchy; we will present alternative models—at different levels of

abstraction—of what is legitimate to do within a conceptual model and how to do it. Thus we are in the realm of meta-meta- and meta-modeling. We will explicitly consider the representation of knowledge in conceptual models, the classes of knowledge that can be modeled, the cognitive aspects of knowledge acquisition and representation in conceptual models, and general methodologies for modeling.

2.2.1 KNOWLEDGE REPRESENTATION IN CONCEPTUAL MODELS.

Methods for representing knowledge in conceptual models have evolved from relatively simple abstractions of data in database design and programming languages to high-level constructs in AI-based expert systems. We will briefly survey this spectrum.

2.2.1.1 Data Models. As a starting point we will consider the classical data model as presented by Brodie (1984). We will note a trend toward greater semantic complexity in data modeling, ultimately fusing descriptive models with procedural *information* using some of the characteristics of programming languages. These characteristics will lead to an examination of specific features in programming languages that support conceptual modeling, such as object-oriented programming. We will then consider the representation of higher level knowledge from the perspective of artificial intelligence.

Data modeling can be seen as a precursor to conceptual modeling; it is the implementation-independent representation of information to be contained in a database that captures the static and dynamic properties needed to support the desired processes (notably transitions and queries). The static properties are defined in a *schema*, which includes all object types, attributes, and static constraints. Dynamic properties are captured in the specifications for transactions, queries, and reports.

The definition of these terms provides a foundation for looking at higher level meta-models of conceptual modeling.

Object: any concept, event, or entity worth recording in a database that *meets information and processing requirements.* (Note the emphasis on the problem context even at this simple level.) Simple objects are irreducible, and are capable of independent existence. Composite objects are composed of two or more objects, and are dependent on their constituents.

Attribute: a single, static property of an object that has no existence independent of the object. Attributes can be used to describe characteristics or relations.

Relation: the mapping of a set of objects that satisfies a given set of constraints. Objects can be related in one-to-many, many-to-one, and many-to-many ways. These are often characterized as networks, hierarchies, tables, and so on.

Operation: an action that changes the state of the database, for instance by adding, deleting, or changing objects/attributes (including relations). Operations also exist for changing the data model itself. Operations can be simple or composite.

Constraints: rules used to define static and dynamic application properties. These rules can constrain objects, attributes, relations, and operations. For instance, a semantic integrity constraint may be used to prevent a database update that would result in an invalid state for the particular data model. Constraints are inherent (suitable for direct representation in the data model to describe the model's basic semantic properties), explicit (defined using a combination of mechanisms provided by the data model to specify structures, relations, and assertions over objects and their properties), and implicit (produced by the interaction of other constraints). Dynamic constraints use pre- and post-conditions to trigger the appropriate execution of operations.

A number of classes of data model exist. A review of these is useful to provide a starting point for the organization of conceptual models.

The *primitive model* consists of simple groupings of uniquely identified objects that can be accessed directly.

Classical models include hierarchic, network, and relational models. Hierarchic and network models organize objects using one-to-many binary relations; they tend to be representational and have a rich potential for inherent constraints. The relational model is based on tuples which can specify many-to-many relation among objects; a set-oriented facility is provided for access and to establish constraints and assertions.

Semantic data models attempt to provide richer concepts and greater expression of meaning. They include:

Extensions to classical models:

Structural model: establishes relational schema with no distinction between objects and relations.

Object-role model: extends the network model by including the concept of a role. An object can play different roles in an application, with different properties for each role (e.g., weapon platform as a target and as an emitter).

Entity-relationship model: combines features of relational and network models, making clear distinctions between objects and relations. Can be depicted as a conceptual network (i.e., objects are nodes and relations are arcs).

Mathematical models: data models with formal notations and definitions of concepts based on either set theory or first-order logic. In the logical view, objects are represented by logical sentences (providing attributes, relationships, and constraints) that are evaluated against a database of facts to answer queries about the model; changes in the application are made through adding or deleting logical sentences, but not necessarily by changing the factual base. This contrasts with approaches in which the model is a schema of object types whose instances are added or deleted from the database to follow changes in the problem environment.

Irreducible data models: models that reduce knowledge to atomic components (which are irreducible). Such a model supports the independent updating of facts and recombination of facts in appropriate ways for a given reasoning strategy; therefore, irreducible models are sometimes seen as having greater modeling precision and flexibility.

Binary-relationship model: an irreducible model that is a restriction of the relational model; relations are binary, rather than n-ary. That is, a relation exists between a single object and attribute pair.

Irreducible relational model: relaxes the binary constraint, but specifies that information will be lost if a relation is decomposed.

Functional data model: the most widely used semantic model. It combines attributes of the relational data model with functional programming. Objects are represented as aggregations of attributes, and relations are functional mappings between objects. The method of functional decomposition lends itself to list processing, the simplicity of which is highly appealing. The model is also irreducible, since each attribute is related to its object by a function.

Finally, *static semantic hierarchy models* integrate relational concepts with attributes of semantic nets. Their capabilities support data abstraction: details are suppressed except for those pertinent to the problem space. The basic activities involved in this process (Bordiga, Mylopoulos, and Wong, 1984) are:

Classification: grouping of entities that share common characteristics into a class over which uniform conditions hold

Aggregation: treating a collection of concepts as a single concept

Generalization: extracting from one or more given classes the description of a more general class that represents the commonalities while suppressing some of the detailed differences

Specialization: introducing detail into the description of a general class to produce a more specialized class

Association: creating a higher level set object from relationships between similar objects

These processes are fundamental to conceptual modeling. They allow model design to proceed in a fashion analogous to the specification of programs by systematic decomposition. Through the addition of programming language capabilities and the strong use of such features as abstract data types, classes, strong typing, and polymorphic types, the semantic hierarchy model is extended to support these processes, becoming a *dynamic semantic hierarchy model*— one that is well suited to modeling the dynamics of the domain as well as its static properties.

2.2.1.2 Abstract Data Types and Object-Oriented Programming.

A key feature of the semantic data models—one which will recur in our review of knowledge representation in conceptual models—is the abstract data type. Ziles (1984) describes a type as a precise characterization of structural or behavioral properties shared by a collection of entities. According to Shaw (1980), abstract data types have emerged from concerns in program language development for information hiding, locality of access, and systematic views of data structures. Their use involves partitioning the program in advance into modules that correspond to major data structures of the final system. The name of the data type and the operations permitted to use it are visible outside of the type definition. The representation of the type in terms of built-in data types (or other defined types) and hidden routines called only from within the module is not visible outside the type definition. Ziles refers to a collection of types, together with a family of operations (such that the operations are closed within the types in the collection) as an algebra. Algebras are seen as the actual building blocks of conceptual models.

The influence of typing and type checking will be apparent in the other modeling methods we will discuss. This influence is illustrative of a trend in data modeling, programming languages, and AI techniques that recognizes (Brodie, 1984) the need for:

Data independence: the separation of logical and physical implementation

Semantic relativism: the ability to view and manipulate data in the way most appropriate for the problem solver

Integration of structure and behavior: the co-specification of both the static and dynamic properties of the system being modeled

Support for data and procedural abstraction: the ability to emphasize relevant details while suppressing irrelevant details

Modeling directness: the ability of the data model to represent the properties of the system being modeled; properties of the application should be inherent constraints of the model

Modeling uniqueness: the representation of a property in only one way in a model; this reduces the number of design choices, subject to requisite flexibility

Precise definition: providing means to precisely define the model, checking consistency, verifying completeness, certifying nonredundancy, and confirming the absence of anomalous results during updates

Understandability: the ability of a model to be understood by its users, which requires economy and independence of concepts, economy of notation, and adequate documentation

Implementability: the ability of the model to be implemented in a robust, efficient manner

The style of knowledge representation seen in dynamic semantic hierarchical models shows movement toward the integration of the descriptive and procedural components of conceptual modeling. This has been illustrated by the use of abstract data types— essentially a programming concept—in such models. This integration is taken farther in

the world of programming languages by object-oriented programming. Experience with object-oriented programming has had a major impact on conceptual modeling using expert systems.

Like an abstract data type, an *object* consists of some private memory and a set of operations (Goldberg and Robson, 1983). The tight coupling of objects and operations differentiates objects from abstract data types (Ziles, 1984). A type is a set of entities to which a set of operations *applies*. Objects communicate through the exchange of *messages*—a request for an object to carry out one of its operations. The set of messages to which an object can respond is referred to as its *interface*. Message ensure the modularity of an object-oriented programming system by specifying what is desired, but not how to do it. Their use is perhaps the major distinction between objects and abstract data types.

Objects are organized into *classes* representing the same kind of system component. The individual objects described by a class are its *instances*. The class definition describes the instance's private memory and how its operations are carried out. A class can be thought of as a collection of attributes and integrity constraints (usually lacking in abstract data types). All instances of a class have the same interface, but have private *instance variables* that support specialization. Thus, an instance *inherits* properties from the hierarchy of classes of which it is a member. Each level in the hierarchy is a progressively more specialized entity. Some object-oriented programming systems allow inheritance from multiple classes (*multiple inheritance*).

Historically, object-oriented programming derives from tools to support simulation, such as Simula and Smalltalk. Given the elegance with which objects can represent the different entities in a dynamic, interactive environment—affecting each other through the consistent mechanism of message passing—it is easy to see why object-oriented programming has strongly influenced conceptual

modeling. It is powerful medium for satisfying the criteria listed earlier.

There are limits to the effectiveness of implementations of object-oriented programming like Smalltalk, which rely on sequential co-routine message passing. Conceptual models that include the representation of self-knowledge and that can deal with open systems and narrative modes of thought (discussed in the next section)—systems which deal with problem-solving methodologies that are only weakly algorithmic—require concurrent message-passing schemes that take advantage of parallelism. Hewitt and deJong (1984) have described a modeling system for such modeling problems. It is based on abstract objects known as *actors*. Actors, like objects, communicate through the exchange of messages. When a message is received, an actor can change its local state, create new actors, and transmit messages. A *serialized actor* is one which can change its state and acts on only one message at a time. Messages are queued for serial actors in order of arrival. An *unserialized actor* is one which never changes its local state; it can process an arbitrary number of messages at the same time. Traditional properties of transactions are implemented by having actors follow the appropriate message protocols (*transaction managers*). Hewitt and deJong state that this meta-model unifies the conceptual basis of both the lambda calculus schools of programming (based on functions and data structures) and the object-oriented schools of programming (based on objects that are separate from procedures). Since the actor model is defined mathematically, it is independent of any particular implementation and provides a consistent foundation for functions, data structures, classes, suspensions, features, objects, procedures, and processes.

2.2.1.3 The AI Perspective. We have seen how data models and programming techniques have evolved to deal with greater semantic complexity, modularity, provability, and ease of maintenance. There is an indistinct boundary between the representa-

tion of knowledge in data models and programming languages, and between these frameworks and AI. Essentially, AI methodologies—and, specifically, expert systems—have evolved to handle situations that require the accumulation and codification of a powerful corpus of knowledge about a problem domain (Waterman, 1985). They operate at a higher level of abstraction—that is, they are capable of more directly supporting conceptual modeling. This is not to say that there is a one-to-one correspondence between expert systems and conceptual models. Expert systems can handle a range of activities, including prediction, diagnosis, interpretation, design, planning, monitoring, debugging, repair, instruction, and control. Yet, each of these activities seems to imply the existence of an underlying conceptual model. The model may be more objective in some applications (e.g., repair, control) and more subjective (cognitive) in others (e.g., design, planning, instruction); but expert systems have evolved diverse techniques for conceptual model representation regardless of their functions. Because of the high-level nature of knowledge in expert systems, their knowledge representation techniques have become fundamental to conceptual modeling.

General methods of problem solving, embodied in procedural languages that employ standard data modeling techniques, greatly restrict the interactive nature of knowledge acquisition and the flexibility of knowledge representation. In addition, experience has shown that the more classes of problem a single, general program can handle, the more poorly it performs on any individual problem. Work in expert systems has therefore concentrated on applying general reasoning paradigms to bodies of comprehensive, high-quality, detailed knowledge specific to a problem. Knowledge representation in expert systems may use some of the techniques previously described, such as objects. Some additional techniques that are unique to AI-based systems are considered next. These techniques represent a continuity of the evolution from simple models of low-order data to the high-level representation of concepts.

2.2.1.4 State-Space Representations.

One of the earliest representation formalisms used for conceptual modeling in AI programs is the state-space representation (Barr and Feigenbaum, 1981). It is not a real representation of "knowledge"; rather, it represents the structure of a problem in terms of the alternatives available at each state of the problem. Solutions can be generated by exhaustively searching the problem state-space. The drawback of this method is that, for interesting problems, the combinatorial explosion of alternatives renders thorough searching impractical. To prune the state-space to reduce the number of branches that must be examined, the system must be able to reason about the state-space. It must, for example, be able to apply heuristics based on emerging experience with the problem space. This implies a need for a higher level of knowledge representation about the world.

2.2.1.5 Formal Logic and Logic Programming.

Perhaps the most obvious choice of formalisms to represent world knowledge is formal logic. In a formal logic, a set of rules of inference is applied to facts which are axiomatic, or already proven to be true, to generate new facts that *must* be true. In addition, statements that can be represented in the logic language can be tested against the body of known facts to determine its truth (subject to the restriction imposed by the incompleteness theorem). Inference rules allow deductions based on the syntax of expressions, regardless of their meaning. The *entailment* (set of inferences that can be drawn) from a set of statements in a logic is completely specified by its rules of inference. Any knowledge base composed of statements within a logic can therefore be kept logically consistent and can be guaranteed to be correct.

Let us briefly examine some of the kinds of logic that have been applied to conceptual modeling (Thayse, 1988). The *propositional*

calculus (also called the *sentential calculus*) deals with sentences that are either true or false. This is the semantic domain of this logic—the only meanings that sentences can have. Propositional calculus has a vocabulary of *propositions* (or *propositional constants*) and *connectives*. The connectives are negation, conjunction, disjunction, implication, and equivalence. Rules are provided for writing combinations of connectives and propositions into well-formed *formulas*. The connectives are *truth functional*: the truth assigned to a formula is known as soon as the truth values of the propositions are known. Truth is assigned to a formula on the basis of an *interpretation function*, or *interpolation*. A formula is *consistent* or *satisfiable* if it has a model—that is, if there is an interpretation that makes it true. A formula is *valid* when it is always interpreted as true, regardless of the interpretation of the propositions it contains.

Although the propositional calculus seems quite powerful as a conceptual modeling tool, it happens that only a small set of correct reasonings can be formalized within it (Thayse, 1988). For example, simple syllogisms cannot be expressed as valid formulae because propositional logic considers propositions to be atomic. On the other hand, the natural languages we use to think about and express our models of the world treat propositions as structured objects whose meaning depends on the meaning of their components. Natural language is also elliptic, having concepts which are bound to specific values by virtue of sentence syntax. Thayse, by analogy to mathematical language, refers to these concepts as *variables*, as opposed to specific entities, which are *constants*.

Handling the greater complexity requires the power of the *predicate calculus* (Clocksin and Mellish, 1984). This logic has an extended vocabulary that includes:

Constant symbol: a symbol standing for a single concept; also an *atom*.

Variable symbol: a symbol that stands for different individuals at different times; a place-holder.

Predicates: statements about variables and/or constants by themselves or in relation to each other.

Compound term: a *function symbol* together with an ordered set of terms as its *arguments*. The number of terms is the *arity*. Functions are actually added to the predicate calculus to make it easier to express knowledge; together with the *equality* predicate, it raises the calculus to a *first-order logic*.

Atomic proposition: a *predicate symbol* with an ordered set of terms as its arguments.

Compound proposition: a set of atomic propositions which may be qualified by *quantifier symbols* connected by *connectives*. The connectives are the same as in propositional logic. The quantifiers are the *existential quantifier* (refers to the existence of some object(s)) and the *universal quantifier* (refers to all of a class of objects).

We will now consider the relationship between .the predicate calculus and *logic programming*, using Prolog as an example. Formulae in the predicate calculus expressed in terms of implication and equivalence can be rewritten in terms of conjunction, disjunction, and negation. In fact, many kinds of transformation are possible without sacrificing expressive power. For this reason, a normalized form, *clausal form*, is used to represent formulae. The construction of clausal form involves:

- Removing implications
- Moving negation inwards
- Skolemizing (removing quantifiers by introducing *Skolem constants* in place of variables introduced by quantifiers; *function symbols* are introduced to remove the universal quantifier)
- Moving quantifiers outward
- Distributing conjunction over disjunction
- Formulating clauses by separating conjunctions (A *clause* is a collection of clauses, each of which is a collection of literals. A literal is either an atomic formula or a negation of an atomic formula.)

As noted earlier, from propositions in a formal logic, other propositions can be derived by applying rules of inference as a set of inference steps. One such rule of inference is the *resolution principle*, which allows the mechanical proof of theorems from axioms; once propositions have been selected to which to apply the resolution principle, valid conclusions are generated automatically. Resolution is designed to work with the clausal form. That is, given two appropriately related clauses, a new clause will be produced. Inference steps deal with three types of clauses:

Denial: ¬A (A is not true)

Assertion: A (A is true)

Implication: $A \leftarrow B_1 \ldots B_N$ (the set of Bs or antecedents imply A, the consequent)

A simple example of resolution is:

if   S1: ¬A

       S2: $A \leftarrow B$

then S: ¬B can be derived from S1 and S2

In resolution terms, the parents S1 and S2 can be resolved to the resolvent, S. When variables occur in the clauses, the process of *unification* is involved; the variables must be *instantiated* to make matching formulae identical.

One approach to using resolution is to derive all possible theorems from a set of propositions and examine them to see if a desired theorem has been proven. Doing so is clearly impractical, and two properties of resolution make it unnecessary. First, resolution is *refutation complete*. That is, if a set of clauses is inconsistent, resolution can derive from them the *empty clause*. Second, resolution is *correct*; it can only derive the empty clause if the set is inconsistent. The empty clause expresses "falseness"—there is no interpretation for the predicates, constants, and functions that make them simultaneously express true propositions. Thus, by taking a set of hypotheses and combining them with clauses representing the negation of conclusions in which we are

interested (*goal statements*), resolution can show that the goal statements are true or false. All that is needed is a strategy for selecting the sequence in which to examine clauses and unify formulae.

Prolog is a logic programming language that provides the resolution mechanism. Prolog operates on Horn clauses, which are a refinement of the more general clausal form. Prolog uses *linear input resolution*. It starts with the goal clause and resolves it with one of the hypotheses to yield a new clause. This is then resolved with another hypothesis to produce a new clause. This process proceeds; at each stage, the clause last obtained is resolved with one of the original hypotheses. A clause whose head matches one of the goals is found, variables are instantiated as needed, the matched goal is removed from the body of goals to be satisfied and the body of the instantiated clause is added. Prolog follows a depth-first strategy to organize the investigation of alternative clauses to satisfy the same goal.

The principal advantage of logic programming languages like Prolog is that the program has declarative rather than procedural semantics. That is, it focuses on specifying what a solution should look like rather than on how to obtain it. It is representing knowledge rather than the details of execution, which are inherent to the language. The programmer formulates a description of the domain and entailments within it; the language supplies a default method of applying it.

The actual implementation of a logical paradigm poses some problems for computation. For reasons of performance and conservation of resources, some control over the theorem-proving process must be utilized. One is that clauses are considered in their textual order in the program. In addition, Prolog introduces a number of structures for controlling the inferencing process and for performing such functions as I/O and data conversion. These methods even allow variation in the axiom set at different points

during the proof. Thus, Prolog deviates from the predicate calculus; some of its functions approximate what higher order logics can provide. It is, however, a simple, practical language with the advantages of declarative semantics and clarity that are expected from logic programming.

Formal logic is useful for addressing what McCarthy and Hayes (1969) termed the epistemological part of modeling (i.e., what kinds of facts are required, how they can be represented, and how conclusions can be drawn from them). Specifically, logic is a natural way to express many kinds of knowledge; it is precise; it is flexible; and it is modular. However, formal logic may be weak in handling the heuristic part of the problem—that is, determining how to use the knowledge stored in the system. As we have seen, logic programming languages like Prolog, FOL, and GOLUX compensate by furnishing control operators while retaining logical precision.

2.2.1.6 Production Systems. *Production systems* (Barr and Feigenbaum, 1981) represent knowledge as sets of rules (productions) in the form of an implication (i.e., a condition-action pair, such as "IF a stoplight is red AND you have stopped THEN a right turn is permitted"). Productions may have associated with them certainty levels on their actions and certainty levels/thresholds on their conditions. The conditions under which a rule applies are made explicit, and interactions among rules are minimized. There are no "calls" among rules.

The data against which the conditional part of a production is compared reside in a *short-term memory buffer*. The condition must be present in a *context data structure* for the production to "fire." These structures can be lists, arrays, or more complex aggregations. Rule testing is controlled by an *interpreter*. When a rule fires, the action may modify short-term memory, assert or retract another rule, or perform an I/O function. During the repeated cycles of condition testing and action that the interpreter

regulates, *conflict resolution* may be necessary. That is, if multiple rules match conditions, a choice must be made as to which rule or rules should be executed. Redundant actions, for example, can be discriminated against.

Production systems have proved useful in controlling the interaction of declarative and procedural knowledge, and have been employed in a number of large, well-known expert systems (e.g., DENDRAL, MYCIN, PROSPECTOR). Recent work has centered on control issues and on self-learning in production systems. A number of desirable features commend production systems for certain types of conceptual models: they are modular (each rule is an independent piece of knowledge), have a uniform knowledge structure, and are natural (in that they describe what to do in certain situations— much as human experts often describe their tasks). The can be very inefficient, however, for the cost of modularity is overhead. Sequences of actions are hard to encode; that is, it is difficult to "chunk" knowledge at different levels. Also, flow of control in the system can be very difficult to determine. Modeling domains for which production systems seem best suited are those in which knowledge is diffuse (composed of many separate facts), processes can be represented as independent actions, knowledge can be separated from its application, and knowledge appreciation is data directed.

2.2.1.7 Semantic Nets. Developed by Quillian (1968) and others, the semantic net was originally created to represent human associative memory. A net is composed of nodes (representing objects, concepts, events, etc.) and arcs (representing interactions and relations). Important associations are made explicit in nets; relevant facts about an object can be inferred by tracing links to related nodes without searching through an entire database. Of particular interest are the arc classes referred to as *isa* and *subset*. These indicate generalization and specialization and establish inheritance in the net.

Using the net formalism, knowledge that is naturally represented in the form of logic statements can be captured. For instance, a *has part* arc connecting the object representing the class "birds" and the class "wings" is equivalent to the proposition: "All birds have wings." If "robin" is connected to the class "bird" via an isa arc, we can infer that a robin has wings. This is equivalent to the remaining statements in a syllogism: "Robins are birds. Therefore robins have wings."

Some types of knowledge are impossible to encode in the basic net structure. For instance, the presence of a property for a bounded period of time cannot be shown by arcs, which have only a binary nature. One solution to this problem, posed by Simmons and Slocum (1972), allows nodes to represent situations and actions as well as classes and instances of objects. Situation nodes can be associated with outgoing arcs called *case frames* that specify the arguments to the situation predicate. · The possibility of inheriting default and expected values is also provided.

Unlike formal logics, semantic nets have no formal semantics—no consistent concept of what a given structure means. Meaning is only provided by the procedures that manipulate the network. A variety of systems have been introduced that use quite different procedures for drawing inferences. An early example is Quillian's *spreading activation model*, used to study memory. More generally, most semantic nets use a matching paradigm. A *fragment* is constructed that represents a desired object or query. It is matched against the network to see if such an object exists. During this process, variables are bound to values required to make a match (like instantiation in Prolog). One such system, SNIFFER (Fikes and Hendrix, 1977), has the power of a theorem prover. It also employs heuristic knowledge expressed by procedures called *selector functions* that describe the order in which network components should be matched.

The semantic net formalism has been elaborated, in particular in combination with the concept of frames (see below). Barr and Feigenbaum note that the formalism cannot be "pushed too far," because of semantic problems and computational limits that arise when networks become large. In particular, it is necessary to consider such questions as:

- What does a node really mean?
- Is there a unique way to represent an idea?
- How is time handled?
- How is degree of belief handled?
- What are the appropriate rules of inheritance for a net?

2.2.1.8 Frames and Scripts. *Frames* were originally proposed by Minksy (1975) for understanding complex human behavior (e.g., natural language dialog, visual perception). Frames were adapted to the representation of sequences of events by Schank and Abelson (1977) in the form of *scripts*. Both frames and scripts are methods for organizing knowledge representation in such a way as to focus attention, facilitate recall, and promote inference.

Frames are compatible with the *structuralist* position in educational psychology: new information is interpreted and organized in terms of what is already known (Piaget, 1972). This facilitates *expectation-driven* processing: looking for data or patterns in data based on the presumed context. A piece of information fits into a *slot;* slots are organized into the context of a larger frame. Like objects, frames are organized to exploit inheritance—particularly as regards the presence of particular slots. Adding new slots, or tightening constraints on inherited slots, accomplishes specialization.

Slots can themselves be complex structures, or pointers to other frames. Slots can be associated with constraints, type checking, defaults, and other *facets. Attached procedures* (also called *active values*) are daemons that respond to updates to a slot to perform a variety of functions (e.g., computing a value).

Thus, procedural knowledge (slot-specific heuristics and triggers) is integrated with declarative knowledge.

A number of systems have been constructed using frame-type representations. Scripts have been used in research. Our own experience shows the power of frame-type structures in representing C^2 systems and their components.

2.2.2 KNOWLEDGE REPRESENTED IN CONCEPTUAL MODELS.

We now turn from how to represent knowledge in conceptual models to what knowledge to represent. The knowledge contained in a conceptual model can range from the relatively objective (e.g., physical measurement or inventories of real-world systems) to the relatively subjective (e.g., creative problem-solving dynamics of human experts). At the objective end of the spectrum, domain experts express a consensus about the scope and meaning of the knowledge represented, although some residual uncertainty (i.e., subjectivity) is always present. At the subjective end, experts tend to have idiosyncratic views of the problem domain. This dimension represents the extent to which there exists a shared conceptual model of the domain. A poorly shared model tends to complicate communication and is accompanied by extremes in expert disagreement. A well shared conceptual model facilitates communication and is accompanied by expert agreement. Differences in methodology can also be associated with the ends of this spectrum.

Problem domains that are more objective lend themselves to mathematical modeling, simulation, statistical methods, analogs, and other concrete mappings onto a modeling matrix. Domains that are more subjective require the techniques associated with artificial intelligence—specifically with expert system design. Our work has dealt with a problem domain, C^2 system design, that essentially resides in the middle ground. While it has objective aspects (e.g., the kill probabilities of specific weapon systems), it also has subjective elements (e.g., the attainment of goals by a specific C^2 system). Even some of the more objective components, such as the components and attributes of a C^2 system, have a subjective component. Soviet and U.S. designers, for example, might chunk the system in different ways. Evidence for this is easily observed in multilingual technical dictionaries. Even though some piece of equipment is described, there is no alingual core meaning. The literature on simulation and mathematical modeling is relatively thorough on the modeling of purely objective problems. We therefore focus on the more subjective elements.

Problem domains also differ in terms of what is known as the *closed world assumption*. This is the assumption that all of the information about the world to be modeled is complete: all and only those relations that can possibly hold among the objects of interest are those represented in the model. In contrast to this, some domains are by nature open; they undergo continual evolution. Such systems tend away from algorithmic and toward problem-solving methods; their distributed components may require significant self-knowledge to accommodate new knowledge that flows into the system.

The use of artificial intelligence methods in connection with conceptual modeling is predicated on the premise that certain classes of phenomena—particularly more open problem domains—are ill represented in an algorithmic way. Such phenomena require heuristic, or discovery, methods that tend to rely on emerging patterns in information representing the system being modeled. Conceptual models developed to support such reasoning must be able to represent more complex structures than are typically found, for instance, in first-order languages.

For example, Kornell (1988) distinguishes the knowledge represented in conceptual models in terms of *formal* versus *narrative* thought processes. Formal thought is concerned with how to know the truth; it seeks closed, well-defined systems, as described

above. Narrative thought, on the other hand, is concerned with the construction of meaning; it seeks open, dynamic systems. The operations of formal thought are described by Kornell as *syntactic:* conjunction and disjunction, deduction and induction, strict implication, instantiation, and idealization—all operating on the formal properties of a referent. The operations of narrative thought are *semantic:* representativeness, plausibility, and cultural appropriateness—all dependent on context. Formal thought may be too constraining for a number of real-world problems—problems where reliance on heuristic knowledge is high. On the other hand, formal thought lends itself to truth maintenance mechanisms which permit the confident application of long chains of inferencing.

Kornell makes the point that problem solving in domains which rely heavily on narrative modes of thought requires conceptual models of expert behavior. The basic ingredients of such models are facts, heuristics, relation-structures, and transformations.

Kornell sees great importance in capturing patterns of reasoning in conceptual models. For example, queuing problem components on the basis of urgency may be required in a given domain. In another, group-and-differentiate or propose-and-revise strategies may predominate. Goals and contextual knowledge are also important, since they have a determining influence on task organization.

Kornell's emphasis is echoed in work on conceptual modeling reported in Noah and Hopf-Weichel (1985). In such domains as military operations and intelligence, narrative thought requires:

Specific knowledge: necessary to interpret specific environmental information, either as discrete elements or as a pattern

Background knowledge: required to interpret specific information within a context

Procedural knowledge: rules used during interpretation (e.g., to construct meaning, draw inferences, make decisions)

Meta-knowledge: knowledge about knowledge

Barr and Feigenbaum (1981) provide the following list:

Objects: We think of the world in terms of facts about objects in the world; therefore, objects, their classes, and their descriptions have to be represented.

Events: Actions and events that affect objects or in which objects engage must be represented; a formalism may also be required for temporal and cause-effect relations.

Performance: Behaviors about *how* to do things—the performance of skills—must be represented.

Meta-knowledge: As in the previous formulation, this is knowledge about what is known. This includes the origin and extent of knowledge, its reliability, and its relative importance. It also includes what the model knows about it own weaknesses, confusability, levels of expertise, etc. Another important category of meta-knowledge is when and how to apply different reasoning strategies, based on the class of problem or state of the solution.

Reasoning strategy: In order to do something it has not been explicitly told to do, a conceptual model must invoke a reasoning strategy (or strategies) that must be encoded within the model, such as:

Formal reasoning: syntactic manipulation of data structures to deduce new ones following prescribed rules of inference

Procedural reasoning: use of simulation

Reasoning by analogy: inference based on similarities among model components

Generalization and abstraction: creation of new knowledge through induction and class comparison

Meta-reasoning: application of meta-knowledge, e.g., selection of inferencing strategies

Johnson, Zaulkeman, and Garber (1988), in their work on the semantic and syntactic analysis of protocols, provide a taxonomy of the kinds of knowledge that can be acquired

from human experts for modeling. Their categories provide a somewhat different insight into the possible referents of conceptual models:

Operations: primitive problem-solving activities that do not depend on a particular context

Episodes: patterns of operations repeated within and across problems

Data cues: operands processed by problem-solving operations

Actions: means of making transitions between problem states

Goals: desired states of the problem

Abilities: capacities to perform actions

Conditions: problem states defined by new data cues

Solutions: end goals

Strategies: permissible ways to move among problem states

The emphasis on different kinds of knowledge in a particular conceptual model based on the narrative paradigm depends on the problem domain. Kornell cites domains typified by:

• Frequent use of analogy

• Frequent recourse to root metaphors

• Correlational (rather than causal) associations

• Problems of recognizing and distinguishing gestalts

• Problems of reconciling conflicting gestalts

Another dimension of problem domains—one particularly important in the context of narrative knowledge—is the extent to which *world knowledge* is involved. World knowledge can be considered knowledge not directly related (or bounded by) the problem domain. A subclass of world knowledge is *commonsense knowledge.* Traditionally, heavy reliance on world knowledge in a domain makes it a poor candidate for knowledge-based problem-solving systems. However, for the purposes of conceptual modeling (in which a representation of a system, rather than a complete, correct solution to a problem, may be the principal goal), world knowledge must

be taken into account. Littman (1988) found that such knowledge can be extremely important.

A theme that emerges is the importance of seeing the conceptual model as a whole, including its problem-solving environment. Woods and Hollangel provide some additional insight into the effect of this holistic view on the contents of conceptual models. Like Kornell, they are concerned with the narrative mode when they describe conceptual modeling in terms of a joint cognitive paradigm based on a problem-driven, rather than a technology-driven, approach. A problem-solving system, such as CMLP, is composed of both automated and human parts. They note that the problem-solving system (from which the underlying conceptual model is inseparable) exceeds the sum of these parts, being determined by its organizational and structural processes. The three elements that make up such a system are the world to be acted on, the agent who acts on the world, and the representation of the world used by the problem-solving agent.

The representation of the world proposed by Woods and Hollnagel (1988) involves a decomposition of the world into a hierarchy of *goals, functions* (set of processes for achieving the goal), and *requirements* (what the processes need to achieve the goal). The *goal-means network* that results from a full problem decomposition is able to capture interactions among goals and interactions among functions. Moving up through the network defines the consequences (and reasons for them) that result from unchecked disturbances in system variables. Moving down the network defines the causal chains and maps the means for correcting disturbances.

In these various meta-models, we have seen a transition from relatively descriptive knowledge to more prescriptive knowledge; from an emphasis on inventories of knowledge to problem-driven knowledge. We have seen that the knowledge in a conceptual model must encompass factual and procedural

dimensions; both a body of facts/rules and a means to draw conclusions from them (i.e., to perform inferencing) are needed. Regardless of the orientation selected for a particular domain, however, there is a common problem. Bylander and Chandrasekaran (1988) refer to this as the *interaction problem*. Briefly put, knowledge representation is strongly affected by the characteristics of the problem and by the inference strategy to be used.

One reason for this is that knowledge must be actively chosen to provide leverage for solving the problem. The knowledge should have high utility; complexity should be reduced. Not everything that the domain expert knows is of equal usefulness. The problem characteristics are salient to this choice. Another reason is that the knowledge will have to be put to use by a mechanism that will use it to draw conclusions—the inference engine. The knowledge must match the capabilities and requirements of this engine. We saw in the previous section that the knowledge-representation formalism depends on the problem and the inferencing technique; we now see that the contents of the knowledge are similarly impacted.

Bylander and Chandrasekaran have reevaluated some generally held beliefs about the knowledge in a conceptual model on the basis of the interaction problem:

• *Knowledge should not necessarily be uniformly represented and controlled in a conceptual model.* This belief denies the interaction problem. There is merit in choosing knowledge (and formalisms) that match the components of the domain.

• *The knowledge base should not be completely separated from the inference engine.* This belief also denies the interaction problem, and historically has led to systems in which inference strategies are implicitly coded in the knowledge base. The separation is artificial.

• *Control knowledge should not be encoded as meta-rules.* Meta-rules address the problem of how to have multiple inferencing strategies in a conceptual model. However,

the separation of control from domain knowledge promotes the view that knowledge can be acquired and represented independent of its application. This implies that different sets of meta-rules can be applied as needed. Given a clear strategy, however, domain knowledge will be adapted to interact with the strategy.

• *The ontology of a domain should not be studied exhaustively before considering how to process it.* Knowledge acquisition should focus on the relevant aspects of the domain for the problem at hand; an exhaustive ontology of the domain is not necessary.

• *Correct reasoning is not a critical goal for knowledge-based systems.* Emphasis on correctness detracts from critical issues, such as the appropriateness of a strategy for a problem. For example, in diagnosis, more may be gained from abduction (assembling composite hypotheses to account for various symptoms) than from computing the uncertainty of each hypothesis to an extended (and artificial) degree of accuracy.

• *Completeness of inference is not a critical goal.* This belief ignores the fact that certain kinds of inferences will be more important than others in a given problem domain.

• *A representation that combines rules, logic, frames, etc., is not necessarily what is needed.* Flexibility is important because it gives the modeler the ability to choose an appropriate paradigm. However, none of the individual representation techniques addresses the interaction problem, and none distinguishes between different types of reasoning.

Having now reviewed both the means of representing knowledge in conceptual models and the kinds of knowledge that can be represented, we will turn to methodologies for acquiring knowledge and constructing models.

2.3 THE NATURE OF CONCEPTUAL MODELS: DOMAIN-DEPENDENT ASPECTS

Applying conceptual modeling to a specific domain requires making a number of

decisions that will define the character of the resulting model. As we have emphasized in the preceding sections, these decisions are based on a number of factors: problem type, use to which the model will be put, available resources (including computer resources, domain experts, knowledge engineering support, and time), contextual factors (social, political, environmental, interfaces to other systems), and end-user characteristics.

For any given set of characteristics describing a problem domain, there will be one or more optimum choices of model configuration. By "configuration" we mean the combination of types of knowledge to be represented (including declarative and procedural categories), knowledge organization, knowledge-representation techniques, I/O and interface properties, and support systems (including hardware, operating system, shells, etc.).

The goals to be met by the configuration (Weiss and Kulikowski, 1984) should be ease of model design (it is generally desirable to get the model running in a short period of time), efficient performance, predictable performance, and capability for empirical testing and validation.

In the previous sections, we have provided an overview of knowledge organization and representation issues. In the section that follows, we will discuss methodologies for developing conceptual models. The focus will be on the general ways that domain-specific factors impact the configuration.

We have repeatedly noted that a conceptual model depends on the type of problem being solved. A taxonomy of problems addressed by expert systems (Waterman, 1986) is a convenient one to adapt to conceptual modeling:

Interpretation: inferring situation descriptions from data (e.g., sensor data)

Prediction: inferring the likely consequences of given situations

Diagnosis: inferring system malfunctions from observable symptoms

Design: configuring objects under constraints

Planning: designing actions

Monitoring: comparing observations to expected outcomes

Debugging: prescribing remedies for diagnosed malfunctions

Repair: executing plans to administer prescribed remedies

Instruction: diagnosing, debugging, and repairing behavior

Control: governing overall system behavior

In conceptual models, the domain being modeled often involves more than one problem type. A C^2 system model, for example, might include facilities for prediction (likely outcomes of force-on-force engagements for particular system configurations), diagnosis (which system characteristics or interactions of characteristics are responsible for real or predicted mission failures), debugging (how to fix the diagnosed problems), and design (how to configure the fixes, given a set of constraints, to accomplish the mission).

Any given problem type has a set of features which further affect the choice of model characteristics. We are unaware of a comprehensive list of these features, but an example (from Waterman) will illustrate the issues. Consider a problem that requires diverse knowledge sources and representations. This suggests a *solution feature* in the form of cooperating subsystems. This solution feature in turn suggests a *tool feature:* a blackboard architecture. In addition to the problem features, there will also be *application features* related to the use of the conceptual model. To carry on with the example, let us assume that the model is to be used in a training system. This suggests that a *system feature* should be the capability of self-modification. This suggests a *tool feature:* rule and control modification facilities.

The accumulation of tool features in the course of this kind of analysis dictates the choice of tool, unless this selection is otherwise constrained. Thus far, we have been

-22-

treating conceptual models as (by definition) implementation independent. As domain specificities enter into the picture, however, tool choice becomes an important issue. As we noted in discussing the interaction problem, this choice will have an effect on knowledge representation and acquisition.

2.4 CONCEPTUAL MODELING METHODOLOGIES

2.4.1 TOP-DOWN METHODOLOGY.
A view of the process of constructing conceptual models (Martin, 1968) which is representative of approaches based on strategies inherited from top-down programming can be summarized as follows:

1. Define the problem. This step includes both a definitive formulation of the problem and a suggested methodology for its solution.

2. Determine information and data requirements. Such questions must be answered as: what information will be necessary? Where will the information be obtained? How will the information be handled?

3. Collect information and data. In conventional modeling, methods used include literature searches, observation, generating artificial data, experimentation, and expert consultation.

4. Create hypotheses about missing or fuzzy data. Assumptions such as these should be periodically and critically reexamined during model construction.

5. Establish a rationale for the model. This is undertaken with consideration for the nature of the real world, the problem, and the tools available to solve the problem. The deterministic and probabilistic elements must be determined, man-machine interactions considered, environmental factors weighed, and functions and pathways specified. On the basis of these data, an approach is created for representing the elements of the model.

6. Define the parameters and variables and weigh the significance and sensitivity of the parameters to the problem.

7. Determine the measures of effectiveness. That is, derive a function that expresses system effectiveness as a function of all parameters and variables. This can be represented analytically or graphically.

8. Determine the approximation procedures. Specify the interactions among the parameters and variables.

9. Validate the model. This is difficult, since the model has not been implemented in software. Such techniques as checking numerical calculations, reverse reasoning, and desk checking are employed.

10. Document the model.

It can be seen that this is a linear sequence of steps predicated on the assumption that what is learned in later stages has no effect on earlier decisions. We have found that this methodological model is inappropriate for conceptual modeling in which expert human knowledge is an essential part. We believe that an appropriate methodology can be drawn from knowledge-based system construction strategies.

2.4.2 EXPLORATORY PROGRAMMING.
As noted by Sheil (1983), the conventional methodology is ideally suited to well-understood problems in which requirements can be specified fully and accurately in advance—simulations, for example. More subjective conceptual models, with a greater component of human judgment, are concerned with problems that cannot be thoroughly specified in advance of design and initial implementation; the expert's experience with the model is a determining factor in its ultimate design.

However desirable the conventional "top-down" methodology may be, it cannot practically be applied to an initial exploration in a field, such as our application of conceptual modeling to C^2.

This situation has much in common with expert system development, in which an approach commonly known as *exploratory programming* is employed. Similar to rapid prototyping, this approach allows design and

-23-

implementation to proceed in parallel; experimentation with the system defines the evolving set of design specifications. Exploratory programming can be adapted to conceptual modeling; it was the prototypical methodology for our modeling work.

The stages of exploratory programming are cyclical and iterative, unlike the conventional approach. Adapted to conceptual model development (from Hayes-Roth et al., 1983), the stages are:

1. *Identification.* This stage involves problem identification, participant identification (e.g., acquisition of the services of domain experts and selection of knowledge engineers and software implementers), resource identification, and goal identification. Note the early role of resources in specifying the problem; we found this to be a major determinant in our study.

2. *Conceptualization.* This stage makes explicit the key concepts and relations identified in the first stage. Concepts and relations elicited from the domain experts or the literature of the domain are typically diagrammed by the knowledge engineers to create a preliminary conceptual model. Data that are documented include data types, strategies and subtasks, hypotheses utilized by the domain expert, object relations, relational structures (e.g., hierarchical, causal, part-whole), solution processes, solution constraints, information flow, and knowledge required to justify solutions.

3. *Formalization.* This stage involves mapping the preliminary conceptual model created in Stage 2 onto formal representations based on the available knowledge representation tools or frameworks. This includes inference rules, control strategies, and data structure contents.

A working prototype system can be used to verify these mappings, and to further refine the model. However, it is the opinion of some authorities that the working prototype should be built with the intent of discarding it once the formalization process is complete. It was our observation that this principle is valid under the conditions of our study. The creative interplay central to the exploratory methodological model argues strongly for separating implementation of the model in a deliverable prototype from its interactive construction.

When experimentation with the model results in an agreed-upon fit of concepts to frameworks, a set of specifications for the conceptual model is prepared.

4. *Implementation.* This stage involves mapping the specifications of the model onto the specific structures, tools, or features associated with the language or system shell of choice. The knowledge is made consistent and compatible and is organized to define a specific information flow, resulting in a deliverable prototype system.

A deliverable prototype can be implemented in one of two ways. The deliverable version can be developed in parallel with model .construction, refining those features which will interface the model to the outside world (the user and other systems or subsystems) and preparing the knowledge representation environment for model integration when its mappings have been adequately verified. The design and implementation of the deliverable prototype must be subjected to rigid software engineering discipline and software validation and verification procedures. The alternative method, which is particularly appropriate if a development shell has been used to construct a working prototype for interactive model construction, is to build on top of the working prototype. This generally requires some restructuring and refinement of the knowledge representation in the working prototype. Our experience was that trying to both construct a model and implement a deliverable computer environment for it (e.g., with a complete user interface) impeded the smooth development of the conceptual model.

5. *Testing.* This stage involves testing the deliverable prototype, first on a limited set of problems and then on a broader set. The domain experts and knowledge engineers are

involved both in problem selection and solution evaluation.

6. *Prototype Revision.* The construction and implementation of a conceptual model require almost constant revision, which may involve concept reformulation, representation redesign, or implementation refinement. The basic measure of the success of revision is convergence on performance. Lack of convergence signals the need for more drastic revision of the architecture or knowledge base.

Much of the refinement will take place using the working prototype. However, even when the model has been sufficiently verified using this medium to be integrated into a deliverable prototype, progressive refinement will still be required.

2.5 APPLICATION OF LOGIC PROGRAMMING TO CONCEPTUAL MODELING

The C^2 design support application developed in this effort illustrates the application of the conceptual modeling problem types·described in Section 2.3, as follows.

Design: The main problem type involved is the configuration of objects (in this case the types, numbers and properties of C^2, sensor, and Weapon elements) under constraints provided by performance and cost requirements.

Interpretation: A supporting problem type for this application is the inference of likely consequences of given situation. In the case of an established baseline, this is the reality of the fielded system. In the case where hypothesized excursions are included, this is a potential system.

Prediction: Another supporting problem type is the inference of the likely consequence of given situations, in the form of attack scenarios and defense configuration and strategy.

Diagnosis: A further supporting problem type is the localization of the cause of poor performance.

Debugging: The prescription of remedies for poor performance is also an important component problem type.

Instruction: Instruction is also present, in the limited form of assistance in (a) using the CMLP system, and (b) identifying further areas of system change. No attempt is made within the CMLP demonstration system to instruct the user in C^2 concepts or design.

The CMLP system and its user are considered as a whole in providing facilities to address these problem types. Without an automated support system, the user would have to solve all these problems and their interaction alone. The CMLP system acts to facilitate the construction and evaluation of solutions to these problems.

The joint man-machine system could be mapped onto the levels explicated in the Gaines (1988) hierarchy (Section 2.2), illustrating its practical application. They are:

1. *Source System.* This is the fundamental level understood by the user; it is basic and implicit, and it provides the sea of shared assumptions underlying the design process. It will be implicitly present in the CMLP system, since a conceptual model will reflect the unstated assumptions of the experts from which it was derived. The user will deem the computerized support incomprehensible and useless if his source system is significantly different from that of the expert relied upon by the knowledge engineer for domain knowledge.

2. *Data System.* This is the level at which all generic and specific baseline data are represented. Since human memory is fallible, the CMLP demonstration system maintains the representation of all data.

3. *Generative System.* This is the level at which hypotheses and predictions are made. In the case of this initial effort, the user generates hypotheses (supported by the data system as presented by the CMLP system). The CMLP system makes predictions about

the effects of the user's hypotheses. (In a future development of the CMLP system, the system would provide some of the hypotheses that must now come from the user.)

4. Structure System. This is the level at which analogies are made. The user may develop an excursion on the basis of excursions made to a different baseline. The computer can provide support by searching through a library of baselines, excursions, and evaluations to offer analogies for the user to consider.

5. Meta System. This is the level at which models are compared. Although the system may provide data to aid the user in this activity, it is beyond the scope of the proof-of-concept effort to automate this activity.

6. Meta-Meta System. This is the level at which the abstract descriptions of the lower levels are productively compared. Facility with this level is the hallmark of the "expert" in a field. Again, this is beyond the scope of this initial effort; the computer system supports by providing visibility into baseline descriptions.

We developed a mapping of the Gaines hierarchy and found it useful, but his mapping into knowledge engineering techniques was amended as detailed above.

At this point, the broad type(s) of problems involved in the application have been identified, and a partition has been drawn between user-provided and machine-provided contributions to their discovery and solutions. The remainder of this subsection develops the design of the computer program by selecting from the alternatives described in Section 2.2.1.

In creating this selection, Bateson's admonition that there is no natural decomposition of a domain is borne in mind. The knowledge representation depends on the inferencing to be performed on it, and vice versa.

The CMLP system requires the following kinds of constructs:

1. Domain Elements. The component elements of the system, and their properties, must be represented. They must also be represented at an appropriate level of detail (or abstraction) for the relationships and activities described below. The representation must allow expansion so that the models may be enhanced and developed. It is a key feature of the development of conceptual models that they are, to a certain extent, open ended.

2. Relationships. The relationships between the component elements of the system must be represented as required for further processing. Examples might include subordination, communication, geographical location, and so on.

3. Activities. The activities in the system must be represented. These activities may include the detection of threat, its engagement, and so on.

4. Contexts. As the design for a C^2 system is developed, some updates will be made to the above elements. Often the user will want to explore some design changes, evaluate them, retract some changes and keep others, and then progress down a new avenue of approach to the problem he is trying to address. The relationship between these design contexts is thus hierarchical, with the user searching through a space of hierarchically organized contexts. Contexts can also be used in solving other problems, such as temporal representations in which the validity of facts is a function of time or of changes in other facts.

5. Interaction. Since this is a design application, in which the user will interact directly with the system, the information must make it easy for the user to understand the state of the model and the consequences of his choices. The system must support the user's memory and should allow the user to deal directly with the problem, rather than dealing with a cumbersome interface. The interface is particularly important for a model that will be widely demonstrated outside a small circle of dedicated and trained users.

The representational techniques discussed in Section 2.2.1 may be applied via logic programming to the CMLP system as follows.

1. Object Representations. Entities within the conceptual model (such as, for the C^2 case, weapon platforms) are represented as instances of the class of all weapon platforms.

Each class has a name and one or more named properties (or *attributes*). Each may be represented as an axiom, thus:

    attribute(C2 Element, Type)
    attribute(C2 Element, Level)
    attribute(C2 Element, Number)
    attribute(C2 Element, Function Status)

These four axioms represent four attributes of the C2 Element class. The attributes are Type, Level, Number, and Function Status.

Each instance of the class has a value associated with each attribute, also represented as axioms. For example:

    value(C2 Element#1, Type, ADOC)
    value(C2 Element#1, Level, 1)
    value(C2 Element#1, Number, 1)
    value(C2 Element#1, Function Status,
        Weapon Status-Manual)

These axioms represent the values of the attributes they name for the first instance of C2 Element (C2 Element#1). For example, the Type is ADOC. Within the class there may be several types (such as SOC and ROC); these would be represented as different instances (say #2 and #3). The Level of the ADOC is 1, the Number is 1 (i.e., there is one of them) and there are several Function Statuses.

When the user adds a new C2 Element to the baseline, the system represents this by adding new axioms to represent the values of the attributes of the new C2 Element. Changing a property of an object (such as the status of a function) is represented as replacing the corresponding axiom with one containing the new value.

It is also important to be able to represent and manipulate knowledge about the representation of the objects themselves; this is called *meta-knowledge.* One type of meta-knowledge applies to values, and represents information such as how many values an object's attribute may have. These may also be represented as axioms, as follows.

    number_of_values(C2 Element, Type 1)
    number_of_values(C2 Element, Function
        Status, many)

A similar mechanism can check to ensure that values are within a legal range or set.

Such axioms would be interpreted during the update process, so that when the user specifies a new value for Type, the old axiom would be replaced. In contrast, when the user supplies a new function status, an axiom is added to those already there.

Further processing may be required for updates, such as when the number of an armament attains some threshold (e.g., O) during an engagement. This requires a rule, thus:

    if_changed(Armament, Number, to(O)):-
        alert(Armament,Number,O)

Such if-changed axioms are sought whenever a value is changed, and the action is executed if the condition is true.

2. Inference. There are several types of inference, which is the computation of the properties of an object on the basis of the properties of (often other) objects:

a. Hierarchical. In hierarchical inference, the properties of objects include those of objects of which they are subclasses. For example, many properties of threats are also properties of threat carriers; among other things, they are both targets. This can be represented as follows:

    type(Threat, Target)
    type(Threat Carrier, Target)

Now when an inference requires instances of Target, the system can return the instances of Threat and the instances of Threat Carrier.

b. Goal Directed. Goal-directed inferencing is needed to determine how a state may be attained. For example, it is used to determine what excursions may impact a given system characteristic. This may be represented in axiomatic form as:

    impact(Threat, Number, Increase,
        Sensor, Number, Increase)

This is interpreted as stating that the impact of an increase in Threat Number may lead to an increase in Sensor Number. To find what may lead to the change in Sensor Number, the axiom set would be used to prove the theorem:

impact(X, Y, Z, Sensor, Number, Increase)

(where X, Y, and Z are variables) and would return the following unifications:

X = Threat
Y = Number
Z = Increase

c. Data Directed. Data-directed inferencing is used to determine the consequences of a situation. For example, it may be used to infer what system characteristics are impacted by a given excursion.

In terms of the above example, the query (theorem to be proved) would be:

impact(Threat, Number, Increase, X, Y, Z)

(where X, Y, and Z are variables) and would return the following unifications:

X = Sensor
Y = Number
Z = Increase

d. Value Based. In value-based inference, a procedure is attached to an attribute that monitors changes to its value and performs activities on that basis. For example, it may be used to generate alerts when a value crosses a threshold. An example has already been given.

3. User Interface. The user interface to conceptual models is often graphically based, since that is a way to present a large amount of complex material grouped in a manner meaningful to the user. Where there is too much data to display, a scrolling menu may be used. Selection of an item in a menu can lead to a new menu being popped up, for a further or more detailed level of selection.

Manipulation of display (or other) devices is extra-logical, but predicates may be provided in Prolog to manipulate them as a side effect of execution. Successful manipulation can return "true," unsuccessful "false."

(As well as organizing textual data, graphical interfaces can also display pictorial data. This capability is not used in the initial CMLP demonstration model.)

The examples given above are simplified for presentation purposes; they are not written in Prolog (although they are close) and they do not reflect actual CMLP demonstration code, for efficiency and other reasons. They do, however, illustrate a general method for representing conceptual models via logic programming.

2.6 LOGIC PROGRAMMING

Logic programming is a relatively new approach to computer program development. The concept of logic programming was developed from the way people think and includes the capability to perform goal-directed inferences for formal proof from a set of facts. In this section we will discuss the history of logic programming, the reasons why we believe it is particularly applicable to conceptual modeling, and the specific advantages we expect to be able to realize on the CMLP project.

2.6.1 HISTORY OF LOGIC PROGRAMMING. Computers are based on binary logic, and they had been in use for only a few years before the "Logic Theorist" was built in 1956 as a part of research at the Rand Corporation and Carnegie Institute of Technology (Newell and Simon, 1956). This program proved theorems in the propositional calculus; both the axioms and the theorems to be proved were taken from the Principia Mathematica (Whitehead and Russell, 1919). Although it was able to prove 38 of 52 theorems, its capabilities in terms of problem reduction and operators required to transform sub-problems were limited. Moreover, it was an experiment aimed at discovering general, domain-independent methods of problem solving, for which the theorems provided examples. It was not a direct assault on the problem of proving theorems by machine.

As the field of artificial intelligence developed, the search for a means to provide a computational predicate calculus remained

an attractive but elusive goal. The breakthrough came when Robinson (1965) developed the resolution principle (described in Section 2.2.1.5) as the basis of a computational theorem prover. Robinson proved that resolution was both complete (would prove all theorems) and sound (would not indicate that a non-theorem was true). Green, a student at MIT, applied this work to the synthesis of conventional programs from their specifications expressed in clausal form (Green, 1969). This work provided, in a rudimentary form, an equivalent of the modern logic interpreter. However, the search space grew exponentially with the size of the set of axioms used to state the problem, so that this was still not a practical technique. Further work by Hewitt (Planner), Sussman (Conniver), McDermott, and others did not succeed in resolving the problem, but generally led American AI researchers away from considering theorem-proving as a potential computational touchstone.

However, theoretical work followed up on some original suggestions made by McCarthy in 1959 (McCarthy, 1968). Techniques for treating logic sentences as program statements were discussed in papers by Hayes (1973) and Sandewall (1973), but the major work in this direction was Kowalski (1974). A thorough explanation in his book *Logic for Problem Solving* (1979) postulated the theoretical underpinnings for practical efforts, but the paper was insufficient to prove the potential.

At about the same time, Colmerauer and others at the University of Marseille built a logic interpreter and named it Prolog for *"Programmation en log*ique." It provided the practical basis for the field of logic programming.

The key differences between the theorem-proving approach and Colmerauer's Prolog were these:

Ordering of Clauses: The predicates are not considered as a set, but in a programmer-defined sequential order.

Restriction to Horn Clauses: There is at most one conclusion that can be drawn per clause.

These two constraints cut down the number of unsuccessful attempts to apply knowledge to such an extent that logic programming became practical.

Colmerauer's interpreter was used for natural language analysis. It quickly led to other implementations, and within a decade to a surprising variety of other uses. At Colmerauer's facility, an implementation was done in Fortran (Battani and Meloni, 1973), and other European researchers sought the means to improve efficiency and make the technology more accessible (Bruynooghe, 1980; Mellish, 1980; Clark and McCabe, 1980). A center of development for Prolog started in Hungary (Szeredi, 1980), where it became a major implementation language, applied to many problems ranging from drug design (Darvas, 1980) to the design of apartment complexes (Markusz, 1980).

But the major development in the West started in 1974 when D. H. D. Warren visited Colmerauer at Marseille-Aix. At that time the term "logic programming" had not yet been coined. Warren had been introduced to logic programming by Kowalski, then at Edinburgh University, U.K., where he was a colleague of Warren. He doubted that an interpreter as simple as Colmerauer's could be effective, but found that he could very easily write a program which defined how a plan could be constructed as a sequence of actions. An action could be appended to the end of a sequence, or it could be inserted in the middle. He found that his program got into an infinite loop unless he used Prolog's clause ordering so that shorter plans were developed before longer ones (Warren, 1986).

Returning to Edinburgh, he and his colleagues quickly developed techniques which improved efficiency to a point at which execution speeds rivaled those of compiled Lisp on comparable hardware (Warren and Periera, 1977). This was very significant since ·

Lisp had received far greater attention. Intriguingly, most of the compiler was written in Prolog, implicitly demonstrating that Prolog was also an efficient language for traditional programming tasks. They also developed an improved surface syntax which has become the de facto standard for Prolog.

At that time Logicon was supporting RADC on an investigation of natural language processing on the Digital Equipment Corporation's PDP-11 computer, designated AN/GYQ-21V. This machine had a 32 KB address space, far too small for useful work from any Lisp environment available at that time. For comparison, the IBM personal computer series has an address space of 640 KB, and each of the CMLP saved states (13 total in the CMLP Demonstration) occupies about 850 KB in the Sun's address space of 4 GB. Early work, using Spitbol, Forth, PDP-11 assembler, and Fortran, had achieved only modest success at implementing the full sophistication of Logicon's theoretical results (Logicon, 1977). In 1978, Logicon acquired a Prolog interpreter for the PDP-11 from Warren's group, and used it for natural language understanding. It was by far the most efficient language for such work with limited resources.

Having proven the potential of Prolog, Logicon organized an invitation-only international workshop on logic programming funded by the National Science Foundation and attended by most of the major Prolog researchers in the world. The intention was to bring the European researchers into close contact with U.S. researchers, to give the language a fair chance in this country. But of the Lisp-based U.S. invitees, only a few (including McCarthy, the progenitor of Lisp) attended. Those who refused apparently failed to recognize that the European approach, though simple, had finessed the key problems that had held up U.S. efforts. Some simply did not understand Prolog. For example, McDermott (1980) dismissed Prolog from consideration in an article that showed that the cut ('!') extra-logical operator was not sufficiently powerful to satisfy complex control requirements. In concentrating on control, he had completely missed the benefits of Prolog (van Emden, 1980).

The Logicon workshop was held in the year following the first international workshop on logic programming, held in Debrecen, Hungary, in 1980, and a few months after an open symposium organized at Syracuse University. It was held a year before the First International Conference on Logic Programming in Marseille. Logic programming symposia have been organized annually in the U.S. and Europe throughout the current decade.

A very early product was an implementation in IBM 370 assembler available from the University of Waterloo, Canada (Roberts, 1980). It is still available, but is completely unportable to other machines. Full-scale commercial development of Prolog interpreters and compilers started in the mid-1980s. In 1984, Warren and his student Byrd joined an effort by Silogic, Inc., to develop a fast Prolog intimately connected to a system providing DBMS capabilities. However, they soon left to help form Quintus Computer Systems, from whom the Prolog software used in the CMLP project was acquired. (Silogic continued with an overwhelmingly ambitious scheme to provide a full natural language interface to their DBMS, and soon folded.) In 1986, the modular Hungarian system M-Prolog (Bendl, Koves, and Szeredi, 1980) was marketed in North America by Logicware, Inc., of Canada. This, although highly portable, had a nonstandard syntax, was not as fast as Quintus's products, and was not as successful. An independently developed European product from BIM also has a nonstandard syntax, but is fast, and its popularity is growing in the U.S. Other products are available, including ALS Prolog (an outgrowth of the continuing research in logic programming at Syracuse University), Prologs from IBM (PSC, Walker), and a variety of implementations for personal computers.

In the U.S., Prolog has received increasing attention since Japan adopted it as the language for its Fifth Generation Project (Warren, 1982). This adoption is of comparable importance to the work of Kowalski, Colmerauer, and Warren.

Appendix C describes Prolog compiler selection for CMLP.

2.6.2 THE CONCEPT OF PROLOG. Prolog was developed to provide a conceptual level of representation that follows directly from logic. It uses the language of predicate calculus that was created by philosophers to investigate valid forms of inference about things and concepts. Philosophers have used predicate calculus to develop simple ways to represent things and their properties. These concepts were adopted by Prolog and are not available in other languages.

Prolog is by its nature declarative. It was designed to allow the user to declare facts that can then be used to support strings of inferences without the necessity for the programmer to define the ordering of the inferencing. Implementations of Prolog do, in fact, employ some interesting sequential approaches in that the comma structures in clause bodies are examined from left to right, and the clause orders within procedures are examined top down. However, a major issue in constructing Prolog tools is to maintain the natural extension to parallel implementations that is allowed in a declarative language.

Prolog allows first-order inferences directly within the language itself. Built-in metalogical predicates provide some capability to support second-order inferences, since terms can be constructed, destroyed, and modified. The two built-in predicates, "assert" and "retract," are nonlogical in that these may destroy the logical nature of the computation by modifying the axiom set in the middle of the proof. There are built-in predicates, such as file manipulation, that provide capability above the natural logic flow of Prolog.

While logic programming provides a strong basis for accomplishing conceptual modeling, pure logic programming cannot be used in its natural form. This is because the utility of conceptual modeling revolves around the ability to make trial changes in the database; this requires an extension of Prolog to prevent destruction of the logical base and, in the process, prevent the changes from being reversible. However, we have been able to overcome this quite easily within CMLP by calling assert and retract within a procedure, allowing them to be undone, and we have shown that the logic programming provides many advantages to the development of conceptual modeling, as discussed in the next section.

2.6.3 ADVANTAGES OF LOGIC PRO-GRAMMING. Logic programming offers many built-in advantages to accomplish implementations of logic models. Major advantages include the following. Section 5.1.2 describes how well we were able to achieve these advantages in the CMLP demonstration system.

1. General Record Structures. Prolog provides no type restrictions on the fields within a clause except as implemented by the programmer. In addition, the programmer may use any number of record types without prior declaration, with any number of fields within the records. This is an important concept in logic programming because it allows the knowledge basis to evolve and provides built-in flexibility for growth.

2. Built-In Pattern Matching. The concept of logic programming is based upon a general and powerful pattern-matching facility that is used in goal-directed inferencing. This means that the usual selector and constructor functions for operating on structure data are not required with Prolog. Procedures for pattern matching are no longer required since Prolog operates primarily in a declarative mode.

Prolog programs are composed of a set of definitions, each of which is an ordered set of Prolog terms. These terms either describe things such an entities or facts, or describe the relationships or rules between things. The

Prolog interpreter works by accepting a query of the form "Does an X exist?" and trying to find a path through its facts to X. The response is Yes if a path is found, No if a path is not found. Note that this does not disprove X, but only establishes that there are insufficient facts within the database to establish X. In this process there are no instructions about what to do or what to do next. The code consists only of descriptions of things that exist and the relationships between them.

3. Multiple Inputs and Outputs. Prolog procedures may have multiple inputs and multiple outputs. Not all inputs need be specified within a procedure execution. There are no fixed commitments to input and output variables, and the arguments of a procedure need not be defined in advance, but may vary from one procedure call to another. This allows a single procedure to act as a constructor, selector, comparator, or a combination of the above functions, depending upon the particular invocation.

4. Both Declarative and Procedural Readings. The language has developed to allow program and data to be identical in form. Therefore the language is completely neutral in terms of procedural vs. declarative representations. The CMLP design process uses procedural steps as well as declarative representations.

5. Intensional and Extensional Data. Prolog also represents intensional and extensional data uniformly. (By an extensional definition we mean one in terms of facts; by intensional, one in terms of rules.) In defining a procedure, the user has the flexibility to decide what these terms are to mean for a set of clauses or whether intensional or extensional data forms are to be used at all.

6. Separation of Logic From Control. Within Prolog, a set of clauses defines the relationships between input and output arguments but does not specify how to compute the outputs from the inputs. The control function is the province of a separately defined deduction mechanism. The Prolog interpreter provides a default for this mechanism of evaluating clauses in a simple top-down, left-to-right fashion, so that for each matching consequent, the first defining clause is used. This provides a clear separation of algorithm and control, despite the fact that there is no distinction made between data and procedure within Prolog.

7. Nondeterminism. The nature of the control sequence discussed above allows procedures to be nondeterministic and generate a set of alternative results. The order in which these nondeterministic results are generated depends on the control regime of the language as discussed above.

8. The Logical Variable. Logic programming allows a unique treatment of programming variables. A variable may remain unbound as long as necessary or convenient. In fact, the variable may not be instantiated until after the computation. Unbound variables may be passed onto procedures and returned from procedures. Unification not only instantiates the variables, but identifies any remaining unbound variable. When one variable receives a value, all variables that are specified by it also receive the value.

This treatment of logical variables incorporates the functions associated with assignments and references in other languages, without the complication of the semantics that are needed in those languages. A second major advantage is that the programmer does not have to make assumptions about the current instantiating of the variable in order to write correct code.

9. Natural Interface to Database Managers. The utilization of logic programming within knowledge base management systems and their interface to traditional relational database management systems is a very active research area. Much of this work focuses on use of logic programming based upon predicate calculus to represent these knowledge variables.

10. Conceptual Level of Representation. Logic programming is based on the language

of predicate calculus to allow valid forms of inferences about things and concepts. This uses the simple mechanisms of predicate calculus to represent things and their properties so that the logic programming language provides the capability to prove statements mathematically. This optimizes implementation of logic variables within the language without having to build a pattern matcher, as is necessary in other languages.

2.7 APPROACHES OTHER THAN LOGIC PROGRAMMING

2.7.1 LISP. Logic programming provides only one of the possible environments for conceptual modeling; indeed, the earliest work on the pertinent issues was done in Lisp (Feigenbaum, 1977) or in a system written in Lisp (see Section 2.7.2). This section describes Lisp and the facilities afforded by Common Lisp, a representative modern Lisp environment.

The Lisp environment has expanded considerably from its first design by McCarthy and his students at MIT in the late 1950s. Originally, Lisp's most important feature was that it was a symbol-processing language in which the code could be used as data by procedures. This alters programs to manipulate its own code: a program could rewrite itself. After Lisp had been in use for some months it was noticed that it was similar to Lambda calculus in that it attaches an argument list to a function. This major feature was not a designed feature of the language.

Lisp was, and remains, a sequential language in which there is an implicit program counter and a default GOTO as the next statement after each statement. Moreover, it is a functional language in which the execution of a statement (or, in Lisp parlance, the valuation of an S-expression), always returns a result.

Lisp is similar to Prolog in that both are symbol-manipulation languages in which a program is capable of rewriting itself. In two other respects Lisp is the opposite of Prolog. Lisp is sequential, while in Prolog procedures to execute are found by pattern matching against a readily expansible repertoire, rather than by naming a single user-defined function. Usually, the equivalent of a Prolog procedure of several clauses would be a single Lisp function containing the equivalent of the Prolog clauses and some conditionals to sort out the selection. More work must then be done to develop a Lisp program. Also, while Lisp is a functional language, Prolog is a declarative language. While a Lisp function returns a result (possibly nil), a Prolog match does not have to succeed completely, returning the logical variable.

The second major difference between Lisp and Prolog is data representation and manipulation. The basic element in Prolog is the term, e.g.,

father(John,Jack)

The basic element in Lisp is the symbol (or atom), with its value, property list, printname, definition and so on. Thus:

(get 'John 'father)

would evaluate to 'Jack if the indicator 'father on 'John's property list had previously been set to 'Jack.

The Prolog term and the Lisp atom can be made to represent many different types of entities or processes. Such freedom of expression is required in the research field of artificial intelligence.

Over the years Lisp has been augmented to provide an integrated environment for editing, inspecting, and debugging Lisp code. Lisp has benefited from much more effort expended by the computer science community than Prolog (on the order of three or four magnitudes).

Common Lisp has recently standardized on an object-oriented extension called CLOS (Keene, 1988). This very flexible system combines features from New Flavors, Loops, Smalltalk, and other object-oriented systems. Method selection is based on generic functions rather than on a message-passing metaphor (in which method selection is determined by the data type of the receiving object). Because generic functions maintain

the same syntax as a standard function call, method selection can be based on any of the arguments in the argument list.

Lisp user interface standards are currently in a state of flux. There are competing interfaces from Lisp to Xwindows (CLX, XCL, etc.), and competing windowing standards (Common Windows [Keene, 1988], CLUE, etc.).

2.7.2 KNOWLEDGE ENGINEERING TOOLS (EXPERT SYSTEM SHELLS). Expert system shells are computer programs designed to provide conceptual modeling facilities. This is in contrast to Lisp and Prolog, which are symbolic programming languages of wide applicability; most of the facilities described above must be built in these languages if they are to be used for conceptual modeling. Knowledge engineering tools differ from language symbolic computation languages such as Lisp and Prolog in providing—or, equally, imposing—an additional layer of structure on top of the language in which they are based. Although Lisp and Prolog represent styles of computation, they are like each other in that the same kinds of application that are conveniently represented in one can be (more or less) conveniently represented in the other. Lisp and Prolog are artificial intelligence tools. Although "definitions" of artificial intelligence abound, it is probably useful to contrast artificial intelligence (named in the late 1950s) as an area of advanced research and knowledge engineering (named in the late 1970s) as an area of practical application.

Knowledge engineering is more constrained than AI; fewer kinds of processes can be represented in a program, but they can be represented with a greater certainty that the process has been faithfully modeled. Knowledge systems development tools are appropriate when the task is to model a domain consisting of objects which have specifiable attributes, and those attributes have values which may be computed from the values of other attributes of other objects in the domain.

What is especially characteristic of knowledge engineering is that the derivation of those new attributes must be as flexible as possible, so that any rule relevant to the current situation, or relevant to new data, must always be available whenever it may be useful. Languages based upon sequential activation of a series of instructions, like Lisp and Fortran, are not inherently suited to this kind of process; they could be called "go to" languages, where the default instruction to go to is the next one. It would not be facetious to remark that knowledge engineering could be said to require, in contrast, a "come from" style of language. In this style, the data transformations required at any point are culled from a collection of potentially useful rules. This is much more like Prolog, where the memory is searched at any given point for rules which could satisfy a given objective (i.e., prove a given theorem).

However, neither Lisp nor Prolog is suitable for knowledge engineering without a superstructure built on top of the basic computational methodology they provide. The collection of attributes, each with their associated values, that represent a given object is represented as a collection of Lisp or Prolog terms. The methods by which the values of an object's attribute may be inferred from others are called "rules" (which are themselves represented with minor extensions of the same representations). What the knowledge system development tool does is to provide its user (the programmer) with the convenience of having to manage the translation from the top level of objects, relations, and inferences into the low level at which the basic computations are performed. This convenience is important: just as it is not cost-effective to have programmers work with machine code writing a bookkeeping program, so it is not cost-effective to have someone doing knowledge engineering in raw Lisp or Prolog. Just as one pays the price of a compiler (and editor) to perform the bookkeeping and make programmers more productive, so one pays the price of a

knowledge system development tool to make knowledge engineers more productive. The initial costs of creating that environment (and debugging, maintaining, and enhancing it) are, essentially, shared by those who purchase it. What makes it cost-effective is that it costs less to buy such a complete and reliable environment than to create one from scratch.

Another characteristic of knowledge engineering programs—particularly expert systems—is that they are highly interactive. They must be able to present visually succinct indications of aspects of their reasoning, and provide the user with the ability not only to examine the line of reasoning, but also to alter it, direct it, or to change the premises and reevaluate. An expert system is meant to be an organic whole, interacting with its user to purposely satisfy a mission requirement.

A knowledge engineering tool is valuable to the extent that it makes creation and validation of such a system easy, and to the extent that it makes it reliable and efficient at run-time. It must be rated as less than valuable to the extent that it makes these things difficult to achieve: if it does not provide a sufficient variety of representations, if it is hard to decide whether tests are successful or not, if there are many special cases (untoward interactions, or even bugs) that burden a programmer's mind, or if it is slow, and hence makes it difficult to work from our limited short-term memory. It can be rated as most valuable, the more facilities it provides, the more salient its visual representation of data, and the less it requires the recall and (error-prone) typing of commands instead of providing recognition and selection.

One knowledge engineering tool is KEE. KEE is a frame-based system. Frames (called "units" in KEE) are structures for organizing knowledge, similar in many ways to the programming language concept called "structures" in PL/1 or C, and "records" in Pascal. Frames are especially useful for representing taxonomies. Relationships between frames in KEE can be member links or subclass links. A member link indicates that a particular object is a member of a class of objects. Subclass links indicate that a class is a specialization of more general class.

Frames have slots, which describe various attributes of the frame. For example, in defining a frame to represent the class of automobiles, we might define slots to represent attributes such as the color, condition, or cost. Frames allow the attributes of objects to be inherited. This is a powerful means of organizing knowledge. If we added a new slot to the Autos definition to represent the fact that an auto typically has four wheels, then when the system discovers that a Toyota Celica is a type of auto, it can automatically assume that the Toyota has four wheels, because this knowledge was inherited from the Auto class. Frames allow the definition of new objects as specializations of other objects that the system already knows about. Thus, you can define a new object merely by describing how it is different from some existing objects. This definition-by-specialization is a powerful technique for hierarchically representing knowledge.

In KEE, rule bases can be defined in addition to frames. In particular, rules can reason about objects that are represented as frames. Frames excel at defining objects and their attributes. Rules and Lisp procedures are better at describing behavior. In KEE frames, rules and Lisp procedures are smoothly integrated. Rule bases and Lisp code can be attached as a slot in a frame. This kind of knowledge allows for a powerful programming paradigm: object-oriented programming.

A method is a Lisp procedure to support the object-oriented paradigm. It is invoked whenever a message is sent to a frame, requesting the value of one of the frame's slots. Methods can also invoke rule bases when necessary. Also, a procedure or rule base can be attached to a frame as an active value; that is, it can be invoked whenever the value of that slot is accessed or changed. Let us consider a resale-value slot of our hypotheti-

cal Autos frame. Since the resale value of a car decreases over time, instead of simply assigning a value to this slot, a Lisp procedure can be attached which takes into account the make and model of the car, as well as its optional accessories, and calculates the current resale value.

KEE is unique in that its rules are normally implemented as frames. That is, a rule is simply a frame with slots for conditions, conclusions, external form, justification for rule, rule author's name, etc. Different rule bases can then be created as classes of rules. This modularity makes it easier to create and debug the rules.

Attributes in KEE are of two basic types. "Own" attributes describe attributes of the class itself. The class Trucks, for example, may have slots for the longest and heaviest known truck. "Member" attributes describe the attributes of each member of the class. Thus, each particular instance of a truck will have a length and weight. More formally, each frame inherits the "Member" attributes of classes of which it is a member, and these attributes become that frame's "Own" attributes.

In KEE, each slot in a frame has both a cardinality and a value class. The cardinality describes how many values that slot may have. For example, the sister slot for the Joe frame may have a cardinality of two. The value class describes the class that each permissible value is in. In this way we can describe the legal values of the decent-professional-man slot as either a doctor or a lawyer but not George.

KEE's rules provide both forward and backward chaining, and are tightly integrated into its truth maintenance system, Keeworlds. Keeworlds is used to represent projections of hypothetical worlds. In other words, whenever one of several possible situations may be occurring, one can maintain each hypothesis inside its own hypothetical world. One can follow separate lines of reasoning inside each world, and probability estimates of the likelihood of each possibility can be maintained. After some arbitrary amount of reasoning, a decision can be made which hypothesis to accept.

High-end system tools such as KEE provide sophisticated rule tracings that show not only which rules are executed and in what order, but also the pattern matchings—that is, the particular information from the domain that was utilized by that rule. By presenting this information in a nicely formatted way, the system can provide explanations of its reasoning that correspond to each step in a logical train of thought that the analyst himself might typically use. The really powerful tools such as KEE go beyond this by keeping detailed information on the justification for each deduction that the system makes. This information is stored in a truth maintenance system. This facility is critical whenever the system must make deductions based on assumptions which later turn out to be faulty. Suppose, for example, that the system makes some deductions about possible tracks based on assumptions about the maximum speed of a MIG aircraft. Suppose later that the system is told that there is a new model of MIG which is significantly faster. Using the truth maintenance system, the system can reexamine all deductions it made based on its faulty assumptions, and can automatically retract faulty conclusions it may have reached.

In summary, the preliminary mapping of KEE features to problem characteristics demonstrates the power and utility of the tool. We visualize utilizing frames to represent hierarchies of concepts such as aircraft characteristics or maneuver categories, using object-oriented techniques to implement the algorithmic portions in a way that makes continual system modification easy, and using rule-based reasoning to emulate the judgmental reasoning of the analyst.

2.7.3 COMPARISON OF ALTERNATIVE IMPLEMENTATION TECHNOLOGIES. In Section 2.6.3 many advantages of logic programming were identified. A disadvantage

of Prolog is its immaturity and lack of development environment. However, the tools that are being developed in Prolog are being designed to supply a large number of built-in capabilities, as shown in Table 2-1. A few of the built-in capabilities are natural from the concept of Prolog, such as the capability of Prolog to support goal-directed inferencing. Other capabilities have been built into Prolog, ProTALK, and ProWINDOWS since they are relative recent developments and are benefiting from experience in development systems for other languages.

The knowledge engineering tool, KEE, provides a tremendous set of built-in capabilities for implementing conceptual models. The cost of these built-in capabilities is some inefficiencies, as well as the cost of procuring KEE. Table 2-1 suggests that there may be a tremendous payoff for development of a knowledge engineering tool for conceptual modeling in Prolog as was realized by the KEE development in Lisp. Section 5.1.4.2 analyzes the potential to apply metaProlog to CMLP; this would provide an equivalent of Keeworlds.

Table 2-1. Comparison of Built-In Capabilities for Candidate Implementations

	Prolog	Prolog Augmented With ProTALK and ProWINDOWS	Lisp	Lisp Augmented With Comm Windows and CLOS	KEE
Object representation					
Hierarchical		X		X	X
Attached procedures (methods)		X			X
Knowledge representation				X	X
Inferencing					
Hierarchical		X		X	X
Goal directed	X	X			X
Data directed					X
Value based					X
Truth maintenance					X
User interface					
Windowing		X		X	X
Graphics tool kit		X			X
Development environment					
Rule breakpoints	X	X			X
Object inspectors					X
Single step and trace	X	X	X	X	X
Hierarchy display					X

89-05- 7a

3. C² MODEL DEVELOPMENT THROUGH CMLP TECHNOLOGY

This section presents the design concept and models for a CMLP tool. Section 3.1 describes the goals of the tool. Section 3.2 presents the overall design and discusses a large number of candidate conceptual representations of C² systems; a subset of these models was selected for implementation in a demonstration system as discussed in Section 4. Sections 3.3 and 3.4 describe user interface, data control, and usage concepts.

3.1 GOALS OF THE MODELS

The original purpose of the CMLP project was to investigate the use of conceptual modeling for a complex domain: the design process for C³I systems. A secondary goal was to identify ways in which conceptual modeling could use logic programming to benefit C³I system designers. We addressed these goals by generating a design concept and models for a tool for use by C² designers. We excluded support to communications and intelligence systems designers to bound the problem and to allow us to work in an unclassified mode.

We began by identifying five classes of potential users of a C² conceptual modeling system, as follows:

1. Technology Development Agency. This class of user, exemplified by RADC CO, performs technology development that may support one or a variety of C² missions. This user needs to understand both the requirements for and payoffs of each technology.

2. Command and Control Acquisition Agency. This user, such as the Air Force Electronic Systems Division or Space Systems Division, develops a specific C² system. This user needs information only about the specific system, but may be able to use data on other systems as a source of concepts for the system of interest.

3. Mission User. This class of user, exemplified by TAC, SAC, or U.S. Space Command, defines the mission element need statement for a particular system. This user typically needs to perform only a high-level analysis.

4. Funding Control Agency. This user, for example AFSC, OSD, JCS, or Congress, is attempting to prioritize the variety of defensive missions and the developments required to support them. This user requires high-level comparative data for C² systems and may have to consider response strategies and defense posture.

5. Operating Agency. This user, such as TAC, SAC, or NORAD (Air Force Space Command), provides day-to-day operation of a C² system and requires a detailed understanding of the system and its performance capability.

From this set we elected to concentrate upon the first two user classes. The broad class of end-mission user may be able to benefit from the CMLP model and was considered in the design process. The funding control agencies require comparisons between systems, which may be feasible through extensions of CMLP technology. The operating agencies require very detailed knowledge of the C² system and were given second priority for the purpose of the design study.

We next identified the following 18 goals that CMLP should fulfill in supporting primary and, to a lesser extent, secondary C² design system users. Italics indicate the users each goal supports. Section 5.2 evaluates achievement of these goals by the CMLP design (presented in Section 3.2).

1. Allow *all users* to analyze performance of defensive system by rule-of-thumb computation.

2. Allow *all users* to analyze performance of defensive system by interfacing with a system simulation.

3. Allow *all users* to analyze C² capability using a goal-scoring approach.

4. Allow *technology developer or C² acquisition agency* to analyze C² system capacity constraints.

5. Allow *all users* to analyze C², communications, weapons, and sensor system life-cycle costs.

6. Assist *all users* by identifying related effects in other models of changes to one portion of the model.

7. Assist *all users* by defining options based on similar situations in the knowledge base.

8. Assist *all users* by evaluating effect of changes based on similar situations in the knowledge base.

9. Assist *all users* by identifying design features incorporated solely to satisfy a requirement whose change is being contemplated.

10. Assist *all users* in organizing and understanding data while entering data into the CMLP knowledge base.

11. Assist *all users* in identifying the data needed to support the design process.

12. Assist *all users* by locating the same or similar data from knowledge bases associated with other baselines.

13. Assist *all users* by performing data consistency checking, either automatically or on demand, against its own knowledge base.

14. Allow *technology developer, C^2 acquisition agency, or mission user* to qualitatively and quantitatively evaluate the effect of a weapons, sensors, communications, or C^2 design change to an existing C^2 system.

15. Allow *technology developer or C^2 acquisition agency* to comparatively evaluate weapons, sensors, communications, or C^2 design changes within a system.

16. Allow *technology developer, C^2 acquisition agency, or mission user* to evaluate expected C^2 system performance.

17. Allow C^2 *acquisition agency* to evaluate stepwise deployment strategies.

18. Allow *funding control agency* to comparatively evaluate design changes across different systems.

To define potential models for the assumed users we next examined the processes and data currently used. We attempted to distinguish between cognitive models (representing a user's perception of a system) and domain-dependent models (more or less realistically representing features of the system). We found considerable inaccuracy in the description of C^2 systems, with the consequence that information entered in a domain-dependent model may, in fact, be inaccurate or based upon prejudices of one particular type of user. The distinction between cognitive and domain-dependent system models is therefore blurred, and we did not consider it to be of significance or utility.

The many alternative conceptual models we identified are summarized in the following two sections. We screened these models on the basis of practicality: whether or not we could identify the contents of the model and appropriately populate it (at least on paper) in order to evaluate its utility. Also, overlap allowed some degree of collection and combination of models. Table 3-1 indicates how the implementing models respond to the 18 design goals and provides references to the sections describing the conceptual and demonstration system model designs. Section 5.2 evaluates the ability of the CMLP design to satisfy each of the 18 goals.

3.2 THE CONCEPTUAL REPRESENTATION OF COMMAND AND CONTROL SYSTEMS

This section describes the models selected for representation of command and control systems. These models are shown in the overall design context in Figure 3-1. Most of them provide the conceptual representation of C^2 systems. Two of them—the analogy development and system sensitivity models—help in using the conceptual representations. The user interface and meta-manager models are described in Section 3.3.

3.2.1 DESCRIPTION MODEL. The description model characterizes the large system in which the C^2 system is embedded. It includes descriptions of the C^2 system, the sensors and weapons, and the threat.

A major design concern was the level of detail needed: detailed information may become a burden to generate, maintain, and use, whereas high-level information may not meet the needs of all users. It therefore appeared important that the description model be constructed to provide an adaptable level of detail. We considered two options. Option A would identify levels of data needed

Table 3-1. Implementing Models for CMLP Design Goals

| Design Goal	Implementing Model	Conceptual Model Design Reference	Demo System Model Design Reference
1. Analyze performance by computation	SWaT	3.2.4	4.1.2
2. Analyze performance by simulation	Simulation interface	3.2.3	*
3. Analyze C^2 performance by goal scoring	Goal attainment	3.2.5	4.1.4
4. Analyze C^2 capacity constraints	Capacity		
5. Analyze system life-cycle costs	Cost	3.2.3	*
6. Identify effect of changes in other parts of system	System sensitivity	3.2.8	4.1.5
7. Define options based on similar situations	Analogy devel	3.2.9	*
8. Evaluate options based on similar situations	Analogy devel	3.2.9	*
9. Identify requirements design dependence	Req & design trace	3.2.2	*
10. Collect data	Description	3.2.1	4.1.1
11. Identify needed data	Description	3.2.1	4.1.1
12. Provide data from knowledge bases	Meta-manager	3.3.2	4.2.2
13. Check data against own knowledge base	Meta-manager	3.3.2	4.2.2
14. Evaluate change to system	All models	–	–
15. Compare changes to system	All models	–	–
16. Evaluate C^2 performance	All models	–	–
17. Evaluate deployment strategies	All models	–	–
18. Compare design changes across systems	All models	–	–

*Not included in demonstration system

89-11-086

to support specific analysis element (the SWaT model [Section 3.2.4] may ignore range effects [no position data], operate on range bins, or accurately model range if positions of sensors, weapons, and threat are provided). Option B would allow the user to select (reduced) information input, and provide warnings and abort analysis where detailed information is not available.

Another design concern was how to model the interactions between elements. In this area we chose to describe the performance of sensors, communications, and weapons based upon their interaction with the threat (considering the impact of the physical environment in the process), and to describe the inter-

action between C^2 and the sensors and weapons in terms of messages.

The interactions to be modeled can become very complex. In air defense, for example, it may be necessary to scramble an aircraft so that the crew can complete the identification of a detected object; thus the crew is functioning as a special form of sensor. If the rules of engagement allow the crew to make the engagement decision, the crew is acting as a C^2 element. After the engagement decision (by the crew or by interaction with higher authority), the same on-board sensor systems (including the crew) used in surveillance may be employed as part of the weapon delivery
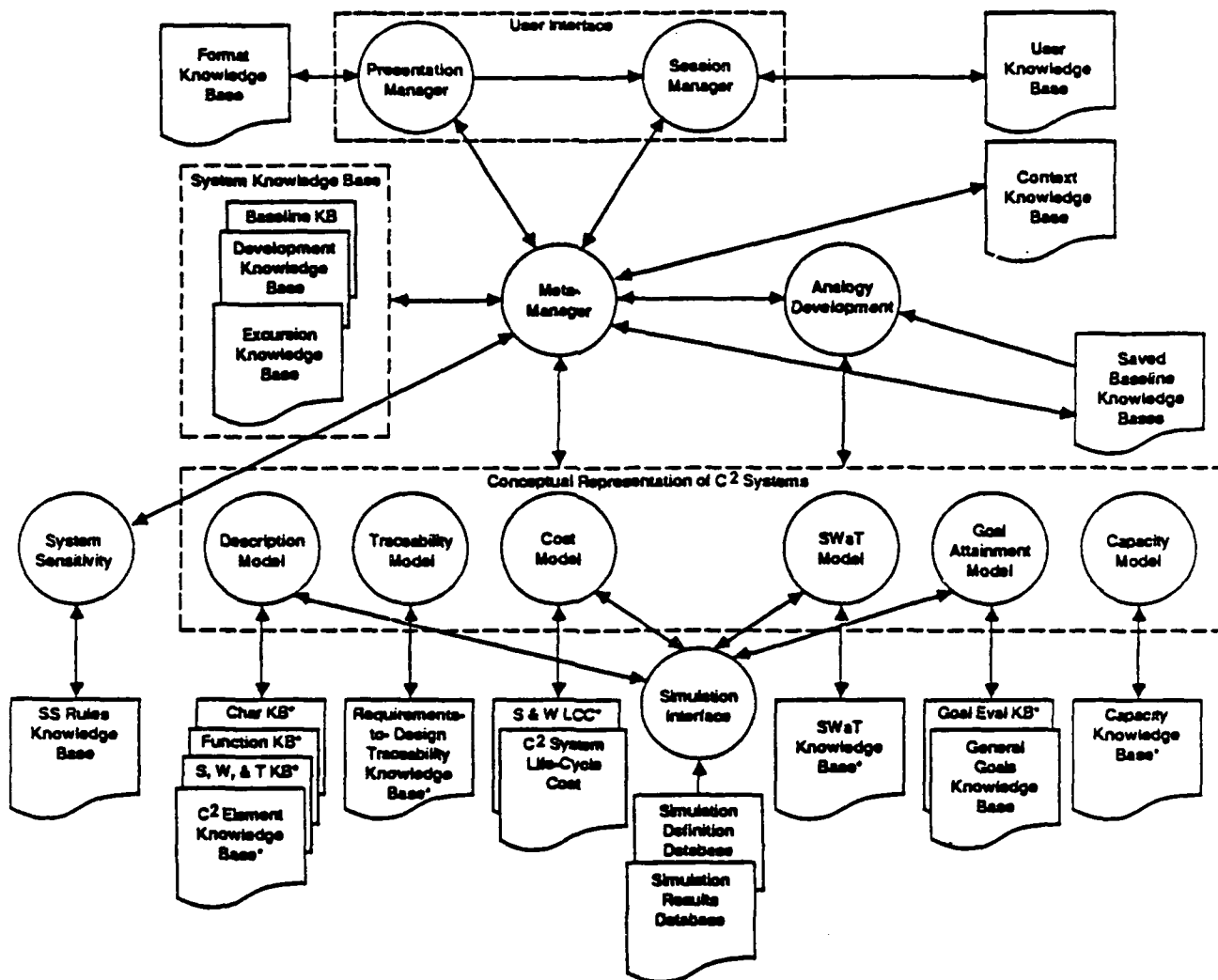
Figure 3-1. CMLP Design Concept

process; hence the crew is now functioning as part of the weapons system.

For the definition model we chose to follow the standard approach for defining C^2 systems, i.e., defining the functions associated with surveillance (or sensing), C^2, and weapon and weapon delivery and implementing these functions with basing on the appropriate platforms.

3.2.1.1 Threat Modeling. Defensive systems are developed to address specific, direct threats and may be characterized as space defense, air defense, surface naval defense, antisubmarine warfare defense, ground force defense, and ballistic missile defense systems.

Counterforce defensive systems are those having the capability to defend by striking the enemy in his own backyard: on airfields before takeoff, in ports, in depots and support echelons, or on the launch pad. We chose to consider these counterforce activities as belonging in the offensive mission area.

In addition to the direct threat elements, another type of threat is that which may have been created to interfere with the defensive system. This threat includes jammers and spoofers, sabotage forces, and other elements that may either destroy or impede the functionality of the defensive systems. This type of threat is of particular importance to C^2 sur-

vivability in that defensive systems tend to have fewer C^2 nodes than either sensors or weapons.

Another characteristic of threat is that a particular threat platform may serve as a launch vehicle for multiple numbers and multiple types of threats: a ballistic missile may deploy MIRVs, or an aircraft may deploy cruise missiles or bombs. In general, threat is described by threat carriers and their armaments. A particular armament (missile, bomb, reentry vehicle, etc.) may have different types of warheads, which can greatly change the nature of defense. Warheads include chemical, nuclear, and conventional and vary in their damage potential (yield).

Also of concern is how the threat may be employed. In defensive system design, the threats to the system itself are typically described in terms of the number of threats believed to be available in the enemy's inventory. To evaluate a defensive system, information on how these threats may be deployed, i.e., a threat scenario, is needed.

The following summary indicates elements that must be included in a threat description. Many of these elements may not be required for a specific level of detail being considered.

- **Threat Bases.** This includes launchers, threat bases, and operating areas for naval forces and may also include storehouses of reconstitution supplies.

- **Threat Carriers.** This may include ships, aircraft, missile launch vehicles, etc.

- **Numbers in Inventory.** This information may be characterized for each base and threat carrier.

- **Location.** This may be by base or by a physical location of a threat carrier in transit.

- **Status.** This may include information on the operational status and potential of the threat element, the number of armaments it still carries, or other pertinent data to the defensive system.

- **Type and Model Number.** This often is an index to further descriptive data such as the characteristics and armament following.

- **Characteristics.** This may include information such as range, velocity, detectability, etc.

- **Armament Type.** This typically describes the armament released from a particular threat carrier such as a bomb, reentry vehicle, cruise missile, air-to-air missile, etc.

- **Armament Lethality.** This describes the probability of damage of the particular armament on the assets being protected by the defensive system.

- **Armament Deployment Ranges.** This describes when the threat platform may deploy a particular armament against the target.

- **Armament Numbers and Capabilities.** This may describe the armament as having multiple warheads, its velocity and range, its capability to detect and close on its own target, and its maneuverability.

- **Other Threats.** This includes the potential for the enemy to employ sabotage, use SPATNAEZ, penetrate security, jam or spoof defensive sensors and weapons, provide forward observers to support threat delivery, etc.

3.2.1.2 Sensor Modeling. Sensors support two principal functions within a defensive system: surveillance and intelligence. We did not explicitly consider sensors used for weapon delivery control unless they could also originate reports generated and forwarded to C^2 to be included in the surveillance fusion process. In most systems individual sensors perform this surveillance function, with their data collected, combined, and analyzed within the C^2 system. Some systems include a group of sensors that operate together and fuse data that are then reported out to the C^2 system for fusion with other sensor groups. The surveillance sensors may have a secondary mission: detecting our own system's weapons and providing a means to report their performance back to the C^2 elements.

Sensors employed by the intelligence function are designed to learn about the nature of the enemy's threat (i.e., collect technical

intelligence) and to collect information that can be used in analyzing his activities and intentions (indications and warnings). It is often convenient to lump intelligence I&W sensors with surveillance sensors.

Sensors may be passive or active. Passive sensors detect emissions/reflections from the target. They include optical sensors that view the target or some of its characteristics and emission sensors that detect the communications or active sensing emissions from a target. Active sensors—the most common is radar—interact with the target and sense the product of the interaction. A ladar system is one in which a laser operates in conjunction with a passive sensor. Ladars and passive optical sensors may operate in the visible, infrared, or ultraviolet ranges.

Sensors may be earth based, space based, or platform based on sea, earth, and air platforms. Sensors are described by their location, number, type and model, performance against different targets, the extent to which performance is affected by environmental or enemy threat elements, sensor sampling rates, sensor reporting rates, and coverage areas and limitations. Performance measures include probability of detection, probability of tracking, probability of discrimination (or appropriately distinguishing between threat elements and friendly forces, and between threat elements), and probability of kill assessment. These performance characteristics nearly always depend on the range to the threat, the environment, and the presence of jammers, as well as on the characteristics of the threat.

3.2.1.3 C³I Modeling. The element under study may require significantly more descriptive detail than the other elements. The description may need to include interactions between elements (such as hierarchy), detailed functionality, basings, figures of merit, rules of engagement, executing and reporting rates, and characteristics of internal choke points.

For the CMLP tool, data obtained through technical intelligence (or the lack thereof) may be considered in terms of the effects on sensor and weapon system performance (or uncertainty concerning performance). This information is not directly treated. I&W intelligence sensors are also not explicitly included, but may be considered by including them in the sensors model.

For analyzing multiple systems, or using one system to analyze another, it is important to describe their C² functionality in similar terms. This requires definition of a common functional organization that is a superset of the functions performed by any C² system included in the analysis. We have not attempted this definition.

Performing survivability analysis will require information on each C² node's resilience to threat. Performing C² design analysis will require design details on operator stations and responsibilities, the numbers and capabilities of processors, the allocations of functions to processors and operators, the rules of engagement employed, the expected accuracy achieved by sensor fusion, and the control steps needed for the particular sensors and weapons employed.

Finally, C² evaluation may also require consideration of defensive approaches other than engaging the threat, for example:

• Passive techniques such as camouflage
• Capability of reconstitution such as repair of a landing strip
• Active countermeasures such as EECM
• Deployment strategies such as mobility, maneuverability, or proliferation
• Physical protection such as use of a hardened facility or radiation-hardened components

3.2.1.4 Weapons Modeling. Performing C² analysis will generally require descriptive information on the nature, basing, and expected performance of all system weapons against each threat type. Most systems will have multiple weapon types. Some weapons will require only minimum C² support, while others will require detailed control procedures during the engagement sequence. Some missions will allow employment of additional weapons if it is determined that a weapon can-

not effectively complete its mission (shoot-fail-shoot) or has missed its target (shoot-look-shoot). Weapons may be described in terms of their current basing, status, velocity, range, susceptibility to attack by the threat, and the numbers and types of armaments included. Armaments may include the shells for a gun, reloads for a missile launcher, missiles on an aircraft interceptor, or capabilities for multiple shots in directed-energy weapons. Most C^2 analyses require the expected lethality of the weapons, and the weapon system conceptual models may need to include the accuracy to which lethality is known.

3.2.1.5 Physical Environment Modeling.

The need for environmental data in C^2 system modeling depends on both the desired modeling detail and the sensitivity of the sensors, weapons, communication, and C^2 elements to environmental effects. Most users of conceptual models can probably ignore environmental considerations. Where the environment is of concern, environments of interest may include nuclear effects, wind, rain, cloud cover, solar and lunar effects, and terrain. For C^2 processes that calculate sensor or weapon performance based upon the weather or other environmental effects, it may also be necessary to include in the environmental model the accuracy to which the environment is known.

3.2.2 REQUIREMENTS AND DESIGN TRACEABILITY MODEL.

The requirements and design traceability model makes it possible for the user to understand the history of a system design. The need is very well illustrated by an example provided by Mr. Sam Di Nitto. Extra-large wheels and an engine inlet covering were provided to allow the F-111 to land on packed sand. This landing requirement was dropped without changing the design, and subsequently the F-111 both experienced accidents due to misperformance of the covering and suffered an extensive penalty due to the size and weight of larger tires.

We note that a model that provides requirements traceability into the design of a par-ticular system is also an effective indicator of system sensitivity (see Section 3.2.8).

3.2.3 COST MODEL.

In theory cost and performance can be used independently to determine the appropriate balance of sensors, weapons, and C^2 capabilities purchased. In practice, however, it is very difficult to separate cost and performance trade studies. The use of conceptual modeling should improve the ability to focus on cost and performance issues and separate one from another.

Cost modeling requires consideration of all elements of life-cycle costs. For example, system design changes may be made solely for their life-cycle cost benefits, without providing any functional (performance) benefits. This situation can arise when, as is often the case, a military system is technologically obsolescent by the time it is deployed. Using such a system requires a source of spare parts, and perhaps a source of new components for proliferation. If the technology for producing these parts is commercially obsolescent, the Government will incur extremely high costs in paying for the maintenance of a manufacturing capability. Of course technology advances may also allow one element to do the work previously accomplished by many. This is particularly true in computers, for which orders of magnitude improvement in capability per dollar occur in just a few years.

3.2.4 SENSORS AND WEAPONS AGAINST THREAT MODEL.

The SWaT model provides a rule-of-thumb estimate of how well the defensive system does when a particular threat is sensed by a particular surveillance system and a certain number and capability of weapons are applied to engage the threat. The performance of most defensive systems depends upon the physical location of the threat, sensors, and weapons, which requires a geographical mapping of all elements for high accuracy. A temporal analysis of the physical locations may be required to achieve a certain accuracy for a particular type of system. At the highest level of complexity, a

SWaT model may in fact represent a simulation of the threat, weapons, and sensors.

We originally included the SWaT model solely to indicate how well the defensive system would perform under ideal conditions, i.e., ignoring the effect of less than perfect C^2. We found that with a minimum amount of effort a rule-of-thumb SWaT model can also be used to aid in evaluating C^2 defensive rules of engagement. This is done by using simple rules to determine threat priorities, and by varying the assignment rules of classes of weapons against classes of threats. If this information is provided at a very high fidelity with consideration of temporal effects, the SWaT model becomes a simulation that includes the simulation of the C^2 elements. Most weapon effectiveness simulation models do not include C^2 effects.

3.2.5 GOAL ATTAINMENT MODEL. The goal attainment model was suggested by the fact that it is part of the human cognitive process to try to estimate how well a particular function is performed. The goal attainment model generically identifies specific things a C^2 system should be able to do; these may then be tailored to a specific C^2 application and evaluated. This type of evaluation is typically done by simulation.

Use of goals in this way in problematic in that there are many goals rather than one measure of effectiveness, or at worst a few such measures. This problem makes comparison very difficult. Work may be needed to define aggregate measures.

3.2.6 CAPACITY MODEL. The physical design of a C^2 system involves a large number of capacity limits in the areas of communications, input and output, computer processing speed and memory capacity, the capability of the human operator to perform manual operations, and the capabilities of the data interchange between the human and the automated C^2 system. Most are limitations in the number of activities that can be done over time, with the consequence that evaluating C^2 system capacity typically requires a temporal analysis.

A capacity model can be created for any system element by modeling the total system capacity and the capacity consumed by each function the element performs. The capacity consumed by a function is usually determined by the number of threat, sensor, and weapon elements involved. When a function is distributed across multiple system elements, it has both location and temporal properties.

Capacity modeling is typically done in the more detailed simulation models.

3.2.7 SIMULATION INTERFACE MODEL. Most detailed analyses of C^2 systems are performed exclusively by simulation. Simulation studies can provide much of the data needed within the CMLP tool, such as specific goal evaluations and calibration for the SWaT model. Simulations may also be used to evaluate system sensitivity, and they may be required to determine the detailed costs.

One use of conceptual modeling may be in defining the minimum number of simulations required, and the elements to be varied from simulation to simulation, to evaluate a particular issue. With the addition of a simulation interface model, a user may initiate an investigation through the CMLP tool. The CMLP tool may identify information needed to complete that investigation, create a series of simulation experiments to evaluate that information, cause the simulation to execute, and then take results and incorporate them within the CMLP evaluation.

3.2.8 SYSTEM SENSITIVITY MODEL. We found that most C^2 systems are evolutionary, and that new concepts are generally defined by considering small performance changes in a variety of different functional areas. The CMLP tool should therefore include a system sensitivity model to help the user identify the effect of a particular change on the other elements represented within the model structure. The model should be capable of assisting the user in two ways. The first would aid in investigating a change to a C^2 system or the defensive system of which it is a part by providing pointers to other areas that need to be investigated (i.e., identifying areas affected by

a discovery that the enemy is manufacturing more threat elements, or by changes in our own weapons, or by changes in C^2 design concepts). The second would aid in finding ways to produce a desired performance improvement by suggesting C^2 or defensive system design changes (i.e., generating and evaluating concepts to increase performance in a specific situation through use of more or better sensors, data fusion in C^2, or faster-response weapons).

3.2.9 ANALOGY DEVELOPMENT MODEL.
The analogy development model allows the user to benefit from data stored in knowledge bases describing other baselines within the CMLP system. We envision two applications of analogy development. The first allows the user to search knowledge bases for system changes to respond to some undesirable condition such as an unsatisfied goal, an inability to counter threat, or limitations in C^2 functionality. The second aids in performing evaluations or securing other data. If the user needs but does not know the cost of maintaining an F-15, he may be able to estimate it based on support costs of other aircraft contained within knowledge bases describing other baselines. If the user needs to evaluate how well a particular goal is fulfilled, he may search other knowledge bases for a similar C^2 system that performs the relevant functions in the same manner. In this case the user would define the relevant functions using the system sensitivity model.

3.3 CMLP DESIGN CONCEPT
The previous section described the models representing or allowing the user to act on the representation of the C^2 system. This section describes what holds the CMLP design together: a very capable user interface and a meta-manager to handle program linkage and the knowledge bases as they are changed under user control.

3.3.1 USER INTERFACE. The CMLP
design concept involves use of a large number of distinct but interrelated models. A typical user session will involve changing the data in two or more models and evaluating the effects

through a variety of other models. To do this the user needs to be able to move from model to model in a straightforward and convenient fashion.

Our CMLP design concept employs a powerful windows package as the basis of the user interface. The package includes a high-level menu bar for identifying the area of the model in which the user wishes to work. A form for that particular model is accessed and populated with data from the knowledge base. The user may accept or modify these data, in most cases moving from the original form to other forms for this purpose or to evaluate the data. To implement the user interface, two logical functions are performed as distributed functions based upon extensive use of the windows package. One function, the presentation manager, accepts mouse clicks and keyboard entries and interprets this input based on the status of the form or menu being displayed and acted upon. The presentation manager also generates new outputs to the user when so commanded by previous inputs.

The second major function is the session manager, which augments the presentation manager by recording in a session state knowledge base the status of the user and system dialog. The session manager maintains a reference of the display structures being used, the current selections being acted on, and the current sets of selections available. It allows the user to personalize the interface to his own requirements via a user knowledge base describing his interests and capabilities. This concept may be used to control user data access and change privileges. The user descriptions may provide a set of information on the types of modeling he is likely to do, his prejudices (e.g., sensitivity rules he wants to have included or ignored), the level of detail needed within the description model, and records of previous activity that he may recall for his own benefit.

3.3.2 META-MANAGER. A meta-manager
provides the interface between the user interface and the system models, controlling the session to make sure that all functions are

executed properly and in the appropriate sequence. It controls such items as what data should be passed, what the detailed state of the current session manager is, and what the state of the session is in terms of the baseline from which the session started, the development baseline, and the new baseline. The meta-manager normally will consist of the following sections:

- Command section to interpret and act upon the information passed from the user interface

- Knowledge section to maintain the state of the knowledge bases being used throughout the system

- Interprocess communication and control section to call the system models and pass data to them

- Context section to maintain the state of the session, including the forms currently open to the user

- Help section to respond to user requests for help by providing information from stored knowledge bases

3.4 CONCEPT FOR CMLP USE

This section describes the steps in CMLP usage (Figure 3-2), assuming that this is the first time the C^2 system of interest is to be considered for a new mission area. The tool is already populated with information on other C^2 systems and with generic information such as generic goals and system sensitivity rules.

The initial step is to provide the tool with the information it needs to deal with this particular system. While the bulk of this information will be provided to the description model, other models may need to be initialized in some way. The capacity model may require identification of the capacity bottlenecks. The system goals should be reviewed and updated for this system, with the user determining whether each generic goal applies to this C^2 system and whether there are specific goals that should be introduced. This system may also have unique features that cause changes in the system sensitivity rules. We would hope that the generic system sensitivity database would be such that most of these changes are

deletions, because this particular system does not include that sensitivity. The basic cost data should be a portion of the description model. However, the cost formulas, such as definition of the maintainability concepts, may require changes in the cost model. The requirements and design traceability model may also use the description model or may require its own inputs. Since each system is likely to have a unique simulation, the simulation interface may have to be developed if it is to be used by CMLP. To populate the description model, the CMLP tool should be able to refer to data stored in other baselines to support this baseline.

The second step is to evaluate the baseline. Performance of the SWaT model will provide some indication that the sensors, weapons, and threat data have been entered correctly. Capacity models may be populated and verified against the data on the capacity-limited elements in the C^2 system. The user will need to perform the initial evaluation, in which process he may draw upon evaluation knowledge databases from other systems.

The remaining steps evaluate possible C^2 changes by manipulating the baseline. In the process of entering the baseline and evaluation data, the user will undoubtedly have generated some ideas for improving the C^2 system. Step 3 takes these ideas and tries them using the CMLP evaluation capability. During this step the user may want to store a new baseline for the system as the result of one or a combination of deltas. A typical delta might be to add a new fusion processor to fuse data from a variety of sensor systems and improve the target knowledge. Before a new baseline can be stored, the user must review areas of change to the baseline using the system sensitivity rules (Step 4).

The user may run out of ideas he wants to try in this C^2 concept design. The analogy development model may be used (Step 5) to identify design improvements for any perceived deficiency in the design, such as an unsatisfied goal. Each change to the C^2 system baseline can be evaluated individually, or
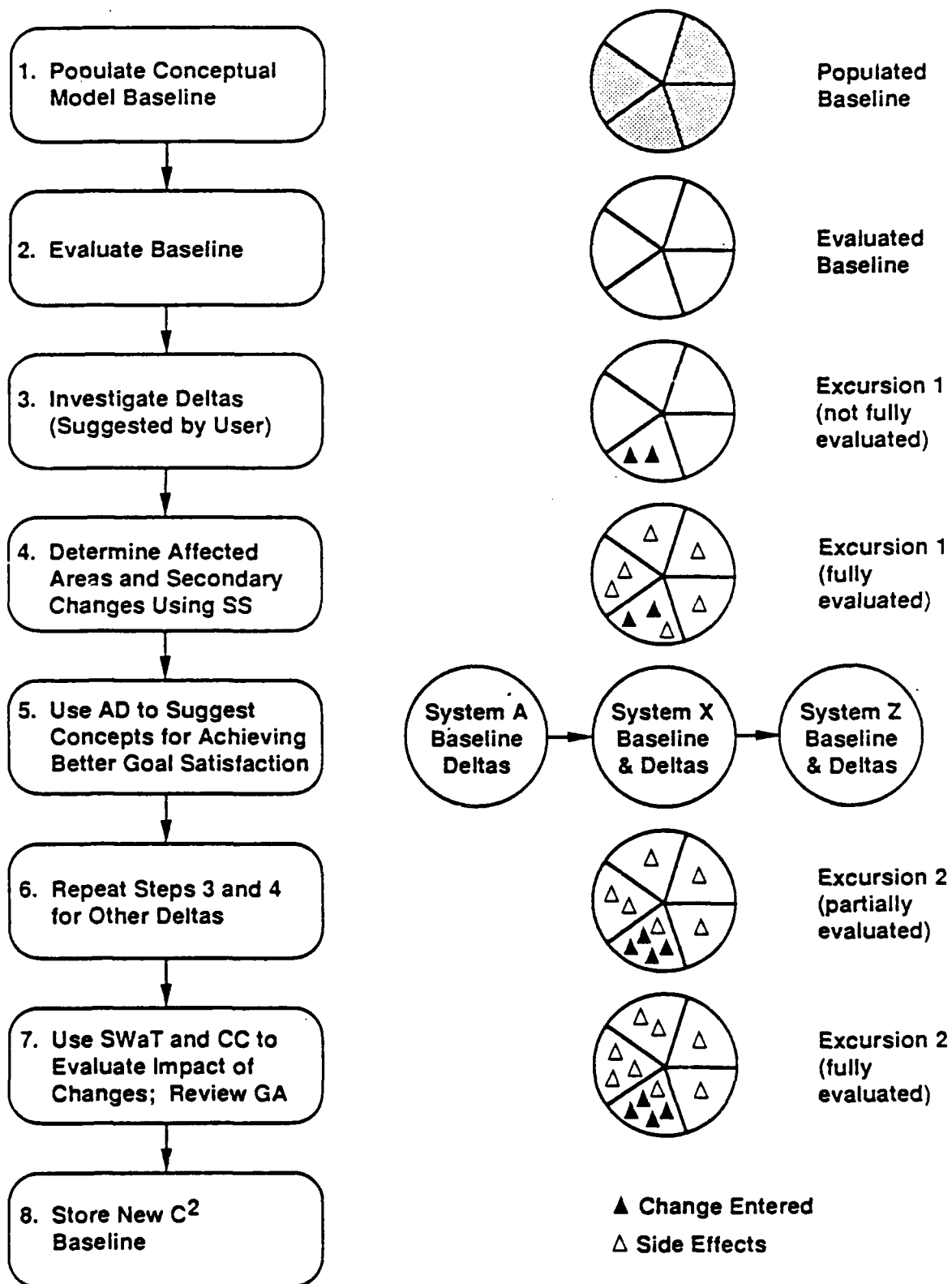
Figure 3-2. Steps in a User's Methodology

all may be combined and evaluated as a group. That evaluation process uses the system sensitivity rules to identify potential areas of change. The cost model is run, and the requirements and design traceability model to identify design features no longer needed. The SWaT and capacity models are run (Step 7) to determine whether there have been significant changes at this level. This may indicate the need for detailed simulation data to evaluate performance after a set of changes. Finally, before storing the baseline (Step 8) the user verifies the goal attainment results.

4. CMLP DEMONSTRATION MODEL

4.1 DEMONSTRATION MODEL DETAILS

This section described the design of the demonstration system, not all of which we achieved in building the demonstration model. The design of the CMLP model was changed substantially in July with a decision to use the ProWINDOWS user interface. This decision was critical. Although the interface is extremely user friendly, it was far more difficult to implement than we had anticipated. A file descriptor problem (described in Section 4.3.4) made it impossible to implement data interchange between Prolog models directly. As a result we lost the rapid-prototyping benefits of Prolog and were unable to get all of the forms integrated and working. However, the as-built model was adequate to demonstrate the principles involved. Any differences between the demonstration system design and the delivered model are noted throughout the section.

The demonstration model design, shown in block diagram form in Figure 4-1, is a subset of the overall design described in Section 3 and shown in Figure 3-1. One omission in the user interface design is the definition of user classes, including their requirements, capability, and outlook (user knowledge base). The demonstration model design does not have the capability of automated personalization for each user class. Also, since models are not implemented to allow comparison to other baselines, some simplifications have been made within the session manager. The meta-manager interface does include recording of the starting or parent baseline for the system, an updated development baseline as the session progresses, and an excursion knowledge base that describes the changes that have taken place in the baseline.

The following sections describe each of the C^2 models within the CMLP demonstration model, starting with the four models used to describe and analyze the C^2 system and ending with the system sensitivity model, which allows the user to identify changes that may need to be made to keep the four models

consistent. Section 4.1.6 then describes the CMLP operational concept in the context of an air defense mission. Screen forms are defined in the User's Manual (Appendix B).

4.1.1 DESCRIPTION MODEL. The description model is a collection of forms and knowledge bases allowing the user to describe a particular C^2 system. The model describes the threat addressed by the overall defensive system, the sensors and weapons portions of that system, and the design concept for the C^2 system providing control for the sensors and weapons.

Geometric considerations are modeled in a simplified form. The user establishes a set of range bins that are used within the threat, sensors, and weapons models to describe the expected performance of these elements.

An overall summary of the data contained within the description model is shown in the baseline display of Figure 4-2, which in this case is populated for an air defense system. Also shown is a summary of the goal evaluation from the goal evaluation model and, at the bottom, a recording of the system description changes made during the session.

The threat and defensive system elements (sensors, weapons, and C^2) include characteristics that affect other C^2 system elements. The values assigned are not directly used by the models developed within the CMLP demonstration system. A list of these characteristics (but not the value, except for C^2) is maintained by the demonstration system and can be used by the system sensitivity model. The list of characteristics for each subsystem is entered through a characteristic developer form as shown in Figure 4-3. This form was designed to allow the user to create, delete, or alter the description of the characteristics. However, the create and delete capabilities do not work in the delivered model. The characteristics incorporated in the demonstration model are listed in Appendix A.

A status of the characteristics of the C^2 system, as the subject of CMLP, is recorded
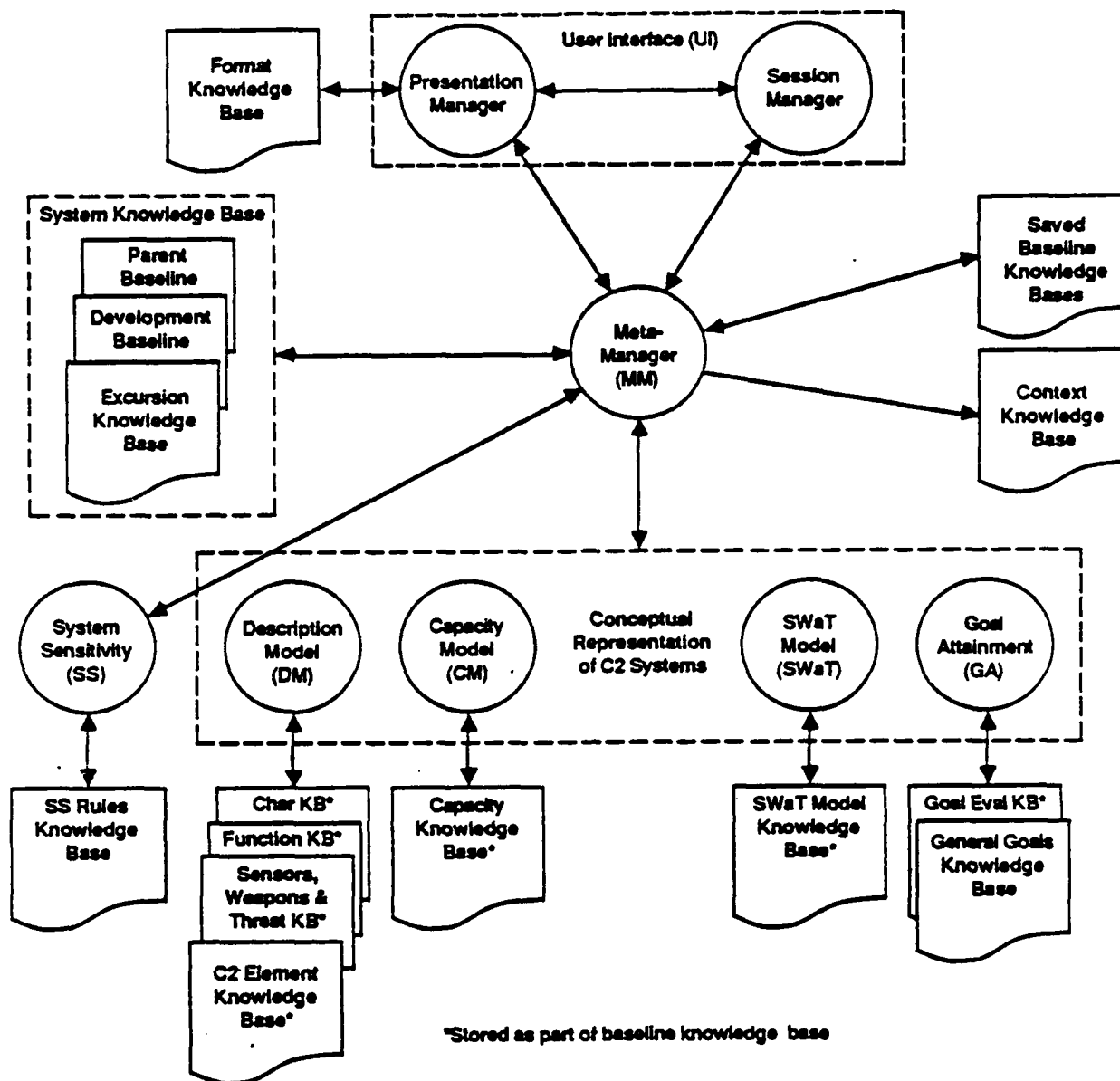
Figure 4-1. CMLP Demonstration Model Design Concept

within the baseline. This is described in the discussion of the C^2 element form (Figure 4-7).

The threat specification form is shown in Figure 4-4. For each threat weapon carrier described in the baseline system, the threat specification display makes accessible the appropriate armaments and their deployment range bins. The example describes the armaments carried on the cruise missile carriers in the enemy inventory, defining the 100 carriers as carrying an average of 10

cruise missiles, including 4 low slow cruise missiles of which 3 are launched in range bin 1, and 6 high fast cruise missiles of which 4 will be deployed from range bin 1. The use of the range bins is discussed further under the SWaT model (Section 4.1.2).

Specification of the sensor system is illustrated in Figure 4-5. (In the delivered software the information shown in the TARGET box will not display.) The user has selected the sensors defined for the longer range bin (1). These sensors include

BASELINE SUMMARY

Author Mark
Date 25 May 1986
System Type ADI
Number 16
Parent System Type ADI
Parent Number 9

C2 ELEMENT	Level	Number
ADOC	1	1
ROC	2	1
SOC	3	6

GENERAL GOAL GROUP	Total	Satis	Partl	Unsat	N/App	N/Sup
System Robustness Go	4	1	1	2	0	0
Surveillance Data Us	6	3	1	1	1	0
System Response Sele	5	0	2	3	0	0
System Timeliness Go	3	2	1	0	0	0
System Authority Con	3	0	2	1	0	0
System Survivability	2	0	0	1	1	0

SENSOR TYPE	Number
Short Range Radar	110
OTH-B	1
Long Range Radar	5
Airborne Radar	2

WEAPON PLATFORM TYPE	Number
Interceptor	300
Interceptor	300
HAWK Launcher	50
Patriot Launcher	50

THREAT CARRIER TYPE	Number
Cruise Missile Submar	48
Cruise Missile Carrie	100
Bomber	80

COMMENT

NOTES

ELEMENT	ASPECT	ORIGINAL	CURRENT
Bomber	Number	80	180
Interceptor	%(Battle)	40	60
HAWK Launcher	%(Battle)	40	60
Enemy Order of Battle Mai	Func Stat	Manual	Automatic
Weapon-Target Assignment	Func Stat	Not Done	Manual
Display Generation	Func Stat	Manual	A&M

Figure 4-2. Baseline Display

forward-based short-range radars and the over-the-horizon backscatter radar. The probability of these radars tracking a threat across range bin 1 has been transformed into a percentage tracked based upon estimate of the enemy attack strategies. The sensor specification form is also used to describe the range bins used in the SWaT model.

Specification of the weapon systems is shown in Figure 4-6. Aircraft interceptors and Patriot ground-to-air missiles may be employed. The armament for each Patriot launcher is shown in the armament column (6 Patriot surface-to-air missiles), along with the expected performance (probability of kill against each target) for the Patriot SAM.

Figure 4-7 illustrates specification of the C^2 system elements. For this particular specification, the hierarchy includes one Air Defense Operation Center, one Regional Operation Center, and six Sector Operation Centers. The functions and characteristics for the ADOC are shown; functions and characteristics may both be scrolled. The bottom window in Figure 4-7 shows functions not done (or not supported) within all of the C^2 nodes.

Within the C^2 element specification the user is provided with buttons to allow

CHARACTERISTIC DEVELOPER
HELP DONE CANCEL LOAD SAVE

ELEMENT
Sensors
Weapons
Threats
C2

CHARACTERISTIC
Number
Location (Primary basing)
Spatial Density
Speed
Detectability
Maneuverability

STORE REVERT NEW DELETE DESELECT

Threat Number:
The number of threats which the enemy has in inventory and are expected to be brought to bear against friendly forces.

a. Threat

CHARACTERISTIC DEVELOPER
HELP DONE CANCEL LOAD SAVE

ELEMENT
Sensors
Weapons
Threats
C2

CHARACTERISTIC
Number
Location
Type
Coverage Volume
Sensitivity
Resolution

STORE REVERT NEW DELETE DESELECT

Sensor Number:
The number of sensors which interface to the C2 system and therefore require data processing by the C2 system.

b. Sensors

CHARACTERISTIC DEVELOPER
HELP DONE CANCEL LOAD SAVE

ELEMENT
Sensors
Weapons
Threats
C2

CHARACTERISTIC
Number
Location
Type
Effective Volume
Sensitivity
Probability of Kill

STORE REVERT NEW DELETE DESELECT

Weapon Number:
The number of weapons which interface to the C2 system and therefore require data processing by the C2 system.

c. Weapons

CHARACTERISTIC DEVELOPER
HELP DONE CANCEL LOAD SAVE

ELEMENT
Sensors
Weapons
Threats
C2

CHARACTERISTIC
Degraded Operations
Mobile
NBC Hardened
Physical Security
Survivable Communications
Back-up Capability

STORE REVERT NEW DELETE DESELECT

Degraded Operations:
The capability of a C2 system to sustain losses in C2 and supporting elements and continue operating at a lower level of capability.

d. Command and Control

Figure 4-3. Characteristic Developer

-54-

complete modification of any element. In the delivered model the ADD and NEW buttons in the element and group browsers do not work. The form was designed to allow the user to add or modify the status assignments within the function box; this provision does not work. The function developer form, shown in Figure 4-8, was design to allow the user to create, modify, or delete functions groups and individual functions within the groups, as well as to change the description functions. This form does not currently work.

The state of each of the characteristics of each C^2 element can be specified on the C^2 element form. The model was designed to allow the characteristics to be generated by the characteristics developer form as discussed earlier and shown in Figure 4-3. This form does not work in the delivered model.

4.1.2 SENSORS AND WEAPONS AGAINST THREAT MODEL. The SWaT model performs a rule-of-thumb calculation on the expected success of the defensive system in

engaging the threat. The demonstration system includes a limited geometric capability by allowing the user to define operations occurring in a number of range bins. The tool does not need to understand the bin definitions, but only the bin attributes. The bins are numbered so that number 1 is the farthest range (at which threat may be detected); processing occurs for that bin first. Armaments are deployed from threat carriers if so specified. Next it is determined which elements of the threat will be detected and, based upon the rules of engagement and weapons assigned, which will be engaged. A percentage of those engaged will be killed according to the weapon probability of kill. The user may allow shoot-look-shoot strate-

Figure 4-4. Threat Specification Form

Figure 4-5. Sensor and Range Specification Form

gies against surviving threat. The threat then proceeds into the next range bin and the process is repeated until all range bins have been completed. The results of the process are reported back as shown in Figure 4-9, with each range bin reported separately before the data is used in the subsequent range bin.

The demonstration system SWaT model allows validation of a variety of rules of engagement but does not include a temporal analysis and does not look at C^2 degradations resulting from delays. A logical extension of the SWaT model is a full temporal system simulation incorporating C^2. In future versions we envision using a SWaT-like calculation for top-level trade studies, supported by detailed simulations to calibrate the high-level SWaT model and to resolve detailed design issues. The SWaT model was designed to run based upon the changed parameters in a trial case using the design excursions summarized on the bottom of the baseline summary form (Figure 4-2). This linkage does not consistently work in the delivered software.

4.1.3 CAPACITY MODEL. A capacity model concept that can apply to any of the elements of the Command and Control system that have limited capability to service a particular function (or set of functions) is needed. To test this concept, the demonstration system has incorporated a limited version of the capacity model. This limited version can be thought of as the capacity available within the total data processing capability for all Command and Control nodes. The total system capability is modeled within the demonstration system, since the demonstration system does not have the capability to geometrically break down the threat, sensors, and weapons as typically occurs in geographically based Command and Control nodes.

The user interface for the capacity model is shown in Figure 4-10. The capacity model has been initialized to a state that describes the capacity being consumed at worst case loading for the baseline system. Inputs include the percentage of capacity being used in the baseline, and a description of how each function using the capacity consumes capacity. The capacity consumption functions are described in terms of arithmetic functions of the numbers of threat elements, sensors in the system, and weapons in the system. For each contributor to capacity usage, the percentage of its current usage is recorded. When changes to the system are being considered, the capacity model can then evaluate the impact of these changes on the capacity limiting item. The capacity model was designed to provide a quick assessment of candidate changes and indicate expanded C^2 capacity needed if the changes were to be implemented. In the delivered software the capacity model does not work.

Figure 4-6. Weapon Specification Form

MODIFY FUNCTION STATUS

GROUP Tracking

FUNCTION Registration

STATUS

☑ Automatic

☑ Manual

☐ Not Supported

☐ Not Done

ACCEPT
CANCEL

C2 ELEMENT SPECIFICATION

HELP   DONE   CANCEL

C2 ELEMENT	LEVEL	NUMBER
ADOC	1	1
ROC	2	1
SOC	3	6

ADD
MODIFY
DELETE
REVERT
STORE

GROUP	FUNCTION	STATUS
Sensor Cntrl	Blanking Control	Manual
Tracking	Sensor Data Acceptance	Automatic
Tracking	Coordinate Xformn	Automatic
Tracking	Registration	Manual
Tracking	Fusion/Correlation	Auto & Man
Tracking	Track Update	Automatic

ADD
MODIFY

CHARACTERISTIC	STATUS
Back-up Capability	Partial
Degraded Operations	Partial
Mobile	No
NBC Hardened	No
Physical Security	Yes
Secure Communications	Yes

MODIFY

Registration:
    The process which applies an algorithm to the received
sensor data to determine positional errors, due to
translation errors and viewing angle, and correct the
errors.

GROUP	FUNCTION	STATUS
Sensor Cntrl	ECCM Control	Not Done
Sensor Cntrl	Radiation Management	Not Done
Sensor Cntrl	Sensor Tasking	Not Done
Telling	Authority Control Arbitration	Not Done
Thrt Assmnt/	Discrimination	Not Done
Threat Eval	Priority Ranking	Not Done

Figure 4-7. C^2 Elements

C2 FUNCTION DEVELOPER

[HELP] [DONE] [CANCEL] [LOAD] [SAVE]

FUNCTION GROUP

Sensor Control
Tracking
Threat Evaluation
Threat Assessment/Id
Weapon Assignment
Weapon Control

FUNCTION

Flight Route Correlation
Route Deviation Alert
IFF/SIF Processing
Geographic Determination
Challenge Processing
Discrimination

[STORE]
[REVERT]
[NEW]
[DELETE]
[DESELECT]

Route Deviation Alert:
The process which monitors tracks on a known flight path and
provides a warning indication should the track deviate from
the flight path.

Figure 4-8. C² Function Reviewer/Developer Form

Range 1 Results

Range 2 Results

Figure 4-9. SWaT Model Form

-58-

```
┌─────────────────────────────────────────────────────────────────┐
│ CAPACITY EVALUATION                                              │
│  ┌─────────┐ ┌─────────┐ ┌─────────┐                            │
│  │  HELP   │ │  DONE   │ │ CANCEL  │                            │
│  └─────────┘ └─────────┘ └─────────┘                            │
│ ┌─────────────────────────────────┬───────────────────────────┐ │
│ │ PROCESSING CAPACITY    Percent  │  ┌─────────┐               │ │
│ │ ─────────────────────  ──────── │  │ MODIFY  │               │ │
│ │ Design Load                 55  │  ┌─────────┐               │ │
│ │ Spare Requirement           25  │  │ REVERT  │               │ │
│ │ ─────────────────────  ──────── │  ┌─────────┐               │ │
│ │                                 │  │  STORE  │               │ │
│ └─────────────────────────────────┴───────────────────────────┘ │
```

PROCESSING CAPACITY	Percent
Design Load	55
Spare Requirement	25

FUNCTION GROUP	Reqrmnt
Sensor Control	7
Tracking	19
Threat Evaluation	8
Thrt Assmnt and Identific	8
Weapon Assignment	26
Weapon Control	12

FUNCTION GROUP	SENSOR	WEAPON	THREAT
Sensor Control	Linear	None	None
Tracking	Linear	None	N Squared
Threat Evaluation	None	None	Linear
Thrt Assmnt and Identific	None	Linear	Linear
Weapon Assignment	None	Linear	N Squared
Weapon Control	None	None	Linear

COMMENTS:
None.

Figure 4-10. Capacity Model Form

4.1.4 GOAL ATTAINMENT MODEL. The C^2 function is the most difficult function to evaluate as there are no natural measures such as probability of kill or probability of track. Simulations are used extensively for determining what the system will do and how that affects overall system performance. As an alternative to a system simulation, a C^2 designer often logically evaluates a C^2 system by determining how well it does each of the functions required. A summary of the evaluation process for ADI is shown in the baseline form (Figure 4-2). The form, designed to allow the user to add, modify, or delete goals and evaluate C^2 performance for each goal, is shown in Figure 4-11. This form currently does not work. A shortcut approach to reviewing or setting the status of each of the

goal uses the goal evaluator form shown in Figure 4-12.

Each of the goals initialized into the database for the CMLP demonstrator is given in Appendix A, Class 5. In initializing a particular baseline model, the user will evaluate the baseline C^2 system against the set of goals. This evaluation can be updated conveniently and easily as design changes are being considered.

4.1.5 SYSTEM SENSITIVITY MODEL. The purpose of the system sensitivity model is to cue the CMLP user to related items so that he may evaluate whether these items are affected by a contemplated change. The model is based upon a set of rules. Eighty-three generic rules are built into the CMLP demonstration database defining both the excursion and the

```
GOAL DEVELOPER

   [ HELP ]   [ DONE ]   [ CANCEL ]   [ LOAD ]   [ SAVE ]

┌ GOAL GROUP ──────────────┐ ┌ GENERAL GOAL ─────────┐  ┌─────────┐
│ ------------------------ │ │ --------------------- │  │  STORE  │
│                          │ │                       │  ├─────────┤
│ System Robustness Goals  │ │ CONSIDER ASSETS AND RISK │ │ REVERT  │
│ Surveillance Data Use Goals │ OPTIMIZE TIME TO INTERCEPT │ ├───────┤
│ System Response Selection Goal │ MULTIPLE WEAPONS COMMITMNTS │ │ NEW  │
│ System Timeliness Goals  │ │ ENVIRNMNTL & GEOMETRC CONDS │ ├──────┤
│ System Authority Control Goals │ GIVE UPDATES TO WEAPONS │ │ DELETE │
│ System Survivability Goals │ │                     │  ├─────────┤
│                          │ │ --------------------- │  │ DESELECT │
└──────────────────────────┘ └───────────────────────┘  └─────────┘

┌ GOAL DESCRIPTION ─────────────────────────────────────────────┐
│ -----------------                                             │
│                                                               │
│ The C2 System shall consider the value of assets              │
│ at risk, degree of risk, probability of defensive             │
│ actions success, and the risk to assets that may              │
│ result from defensive actions.                                │
│                                                               │
└───────────────────────────────────────────────────────────────┘

┌ ────────────────────────┐ ┌ CRITERION FOR DEGREE OF GOAL SATISFACTION ┐
│ □ Satisfactory          │ │ ----------------------------------------- │
│                         │ │                                           │
│ ☑ Partially Satisfactory│ │ PARTIALLY SATISFACTORY - The system utilizes │
│                         │ │ limited information and rules of thumb for │
│ □ Unsatisfactory        │ │ assigning weapons to specific threats or makes │
│                         │ │ unsupported assumptions in the nature of the threat │
│ □ Not Supported         │ │ type for performing assignment.           │
│                         │ │                                           │
│ □ Not Applicable        │ │                                           │
└─────────────────────────┘ └───────────────────────────────────────────┘

┌ AFFECTS ──────────────────────────────────────────────────────┐
│ SWeT Model ○ Yes  Capacity Model ○ No   Geography Model ○ No   │
└───────────────────────────────────────────────────────────────┘

┌ APPLICABLE FUNCTIONS ────┐ ┌──────────────┐
│ ------------------------ │ │  [ ADD ]     │
│                          │ │  [ DETAILS ] │
│ Raid Composition         │ │  [ DELETE ]  │
│ Strength                 │ │              │
│ Priority Ranking         │ │              │
│ Weapon Probability of Kill │ │            │
│ Weapon-Target Assignment │ │              │
│ ------------------------ │ │              │
└──────────────────────────┴─┴──────────────┘

┌ ADDITIONAL REMARKS ───────────────────────────────────────────┐
│ ------------------                                            │
│                                                               │
│                                                               │
│                                                               │
│                                                               │
└───────────────────────────────────────────────────────────────┘
```

Figure 4-11. C² Evaluation Goals

```
GOAL EVALUATOR

   [ HELP ]   [ DONE ]   [ CANCEL ]

┌ GOAL GROUP ──── GENERAL GOAL ──────────── EVAL ..TION ──┐ ┌─────────┐
│ ------------    ----------------------    ------------  │ │ MODIFY  │
│                                                        │ ├─────────┤
│ System Robustness  Maximum Case Attack    Unsatisfactory │ │ DETAILS │
│ System Robustness  Processing for all Regions  Satisfactory │ ├──────┤
│ System Robustness  Conserve Defensive Resources  Unsatisfactory │ │ STORE │
│ System Robustness  Expend Resources Proportionall Partly Satis │ ├──────┤
│ Surveill Data Use  Use all Data Received  Partly Satis │ │ REVERT │
│ Surveill Data Use  Not Rely on Deniable Data  Unsatisfactory │ └─────────┘
└────────────────────────────────────────────────────────┘
```

Figure 4-12. Goal Evaluator Form

impact in terms of element, characteristic, and change; see Class 6 in Appendix A for details. The knowledge base contains the rationale and notes for users as also included in Appendix A, as well as the goals (Class 5 in Appendix A) that may be affected by the excursion. For example, Rule 15 is:

Excursion Element: Threats
Excursion Characteristic: Number
Excursion Change: Increase
Impact Element: Weapons
Impact Characteristic: Number
Impact Change: Increase
Rationale: Ensure that the weapon systems are not overwhelmed by a numerically superior force.
Notes: In some cases, the weapon systems may be superior even though they are smaller in numbers. Adding more weapons to the battle could increase the loads on the C^2 facility.
Applicable Goals: [Maximum Case Attack, Conserve Defensive Resources, Expend Resources Proportionally]

The user identifies an excursion (nominally a change he is considering or has made) and the program searches the rule knowledge base to identify areas of potential change. The user may, at his discretion, make adjustments to the model in the area identified. The design intent was to keep a record of all excursions being considered and display this information on the bottom element of the baseline display (Figure 4-2). This capability does not work in the delivered demonstration model.

The mechanization for this cueing process can also be used to suggest design changes. For example, if improved tracking is needed by C^2, Rules 39 and 41 suggest increasing either sensor coverage volume (allowing more sensors to see a particular object) or sensor sensitivity (achieving better object resolution).

The system sensitivity model has the capability to retrieve rules from the rules base by specifying either the antecedent or consequence of a rule. This allows the user to accomplish inference from change to impact or from impact to change.

Figure 4-13 shows a developer form for system sensitivity rules. The buttons do not work on this form.

4.1.6 OPERATIONAL CONCEPT EXAMPLE (AIR DEFENSE). The concept for the CMLP demonstration model is illustrated for an air defense system in Figure 4-14, which presents an example populated baseline. A baseline is placed under version control using version descriptions as shown in the upper left. It includes the number of each type of threat, sensors, weapon platforms, and weapons as shown across the top. The characteristics of the defensive system are recorded (characteristics are defined in Appendix A, Class 1) using the element specification form. The figure shows the user's selection in terms of Yes, No, or Partial. The C^2 elements are described as shown near the center of the figure, as well as the functions they perform (see Appendix A, Class 2) and whether they are performed Manually, Automatically, or are Not Done or Not Supported (i.e., the sensing or engagement capabilities of the defensive system are insufficient to support the function). The population of a C^2 function database for air defense is shown on the second page of Figure 4-14.

The performance of each C^2 function is rated through goal evaluation as shown at the bottom of the first page of the figure. There are 29 possible goals (Appendix A, Class 4) making up 8 goal evaluation groups (Appendix A, Class 5). Allowable evaluations are Satisfactory, Partially Satisfactory, Unsatisfactory, Not Supported (i.e., the sensors or weapons do not provide enough capability to allow the C^2 system to achieve this goal), or Not Applicable (i.e., this goal does not pertain to this particular defensive system). Also shown at the bottom of the first page is population of the capacity model to allow monitoring of possible C^2 bottlenecks as design changes are considered. Indicated

```
┌─────────────────────────────────────────────────────────────────────────────┐
│ SUBSYSTEM SENSITIVITY RULE DEVELOPER                                         │
│ ┌──────┐ ┌──────┐ ┌────────┐ ┌──────┐ ┌──────┐                              │
│ │ HELP │ │ DONE │ │ CANCEL │ │ LOAD │ │ SAVE │                              │
│ └──────┘ └──────┘ └────────┘ └──────┘ └──────┘                              │
│ ┌EXCURSION ELEMENT──────┐ ┌CHARACTERISTIC──────────┐ ☐ Increase  ┌──ADD───┐ │
│ │ ------------------    │ │ ----------------------  │            ┌─DELETE─┐ │
│ │ Sensors               │ │ Number                 │ ☑ Decrease ┌─STORE──┐ │
│ │ Weapons               │ │ Location (Primary basing)│           ┌─REVERT─┐ │
│ │ Threats               │ │ Spatial Density        │ ☐ Different           │
│ │ C2                    │ │ Speed                  │                       │
│ │ ------------------    │ │ Detectability          │ ☐ Unchanged          │
│ │                       │ │ Maneuverability        │                       │
│ ├IMPACT ELEMENT─────────┤ ├CHARACTERISTIC──────────┤ ☑ Increase  ┌──FIND──┐ │
│ │ ------------------    │ │ ----------------------  │            ┌DESELECT┐ │
│ │ Sensors               │ │ Number                 │ ☐ Decrease           │
│ │ Weapons               │ │ Location               │                       │
│ │ Threats               │ │ Type                   │ ☐ Different          │
│ │ C2                    │ │ Coverage Volume        │                       │
│ │ ------------------    │ │ Sensitivity            │ ☐ Unchanged          │
│ │                       │ │ Resolution             │                       │
│ ├RATIONALE──────────────┴────────────────────────────────────────────────┤ │
│ │ --------                                                                 │ │
│ │ In order to provide detection capability at the ranges                   │ │
│ │ comparable to that which is achievable with threats of normal            │ │
│ │ sensor signature or ECM capability, more sensitive sensors are           │ │
│ │ necessary to detect the threat.                                          │ │
│ │ --------                                                                  │ │
│ ├NOTES──────────────────────────────────────────────────────────────────┤ │
│ │ -----                                                                    │ │
│ │ Increasing the sensitivity of fielded systems may not be                 │ │
│ │ practical or possible and a new sensor design concept may be.            │ │
│ │ necessary.                                                               │ │
│ │ -----                                                                    │ │
│ ├APPLICABLE GOALS───────────────────────────────────────┬────────────────┤ │
│ │ ---------------------------------------------------    │  ┌───ADD────┐   │ │
│ │ USE ALL DATA RECEIVED                                  │  ┌─DETAILS──┐   │ │
│ │ CORRELATE & FUSE DATA                                  │  ┌─DELETE───┐   │ │
│ │ CONTROL RESOURCES                                      │                │ │
│ │ SEND DATA & DIRCTVES IN SUFFCT TIME                    │                │ │
│ │ ---------------------------------------------------    │                │ │
│ └───────────────────────────────────────────────────────┴────────────────┘ │
└─────────────────────────────────────────────────────────────────────────────┘
```

Figure 4-13. System Sensitivity Rules Form

nearer the center is population of the SWaT model, for which the controls and weapons are shown in abbreviated form. The system sensitivity model is not shown because this baseline did not tailor the nominal sensitivity rules (Appendix A, Class 6) for this air defense system.

4.2 EXPLOITATION OF LOGIC PROGRAMMING

4.2.1 IMPLEMENTATION OF CMLP DEMONSTRATION SYSTEM. An overriding consideration for CMLP was the quality of the demonstration, so as a practical matter the most conservative development plan was
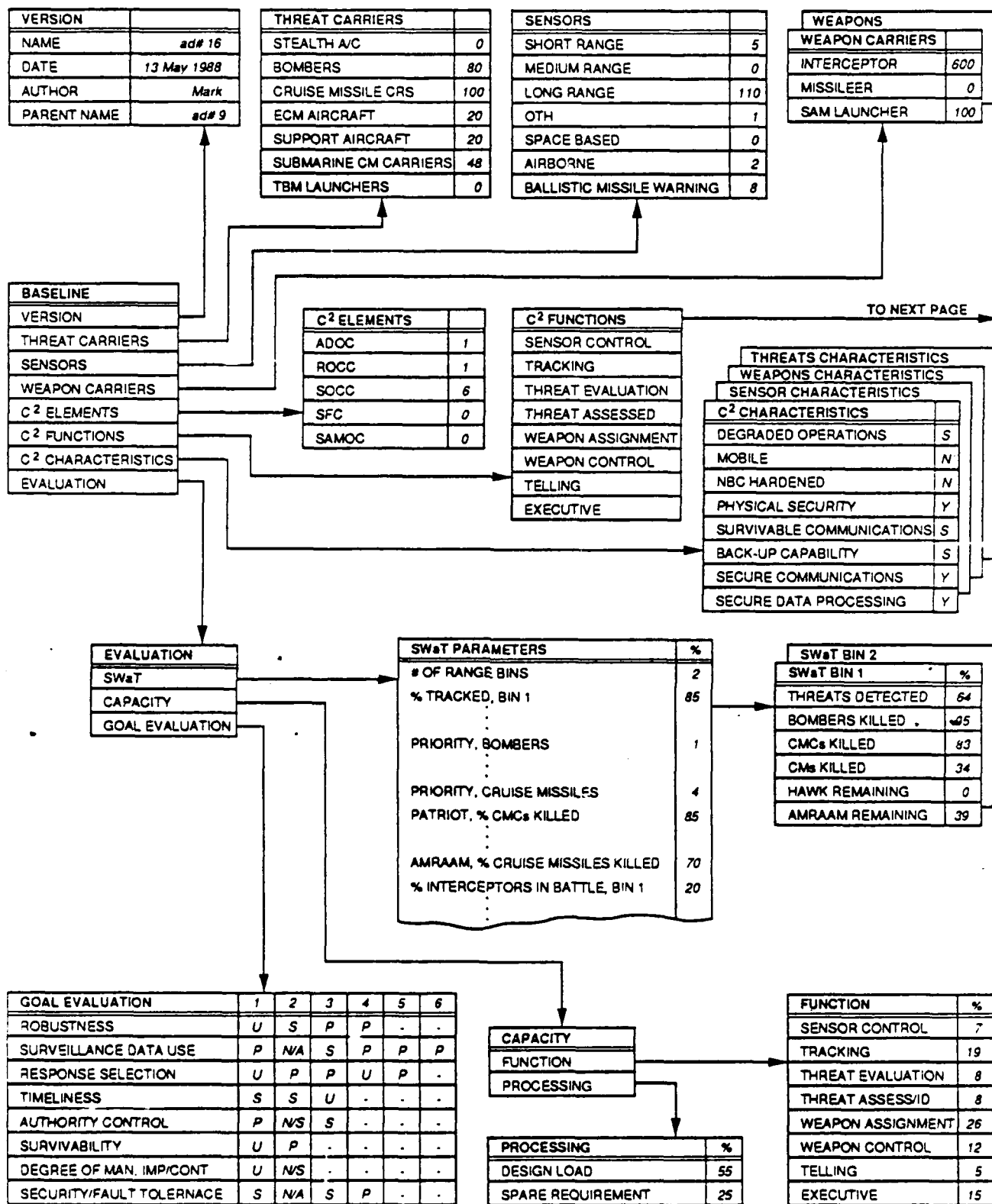
VERSION

NAME	ad# 16
DATE	13 May 1988
AUTHOR	Mark
PARENT NAME	ad# 9

THREAT CARRIERS

STEALTH A/C	0
BOMBERS	80
CRUISE MISSILE CRS	100
ECM AIRCRAFT	20
SUPPORT AIRCRAFT	20
SUBMARINE CM CARRIERS	48
TBM LAUNCHERS	0

SENSORS

SHORT RANGE	5
MEDIUM RANGE	0
LONG RANGE	110
OTH	1
SPACE BASED	0
AIRBORNE	2
BALLISTIC MISSILE WARNING	8

WEAPONS

WEAPON CARRIERS

INTERCEPTOR	600
MISSILEER	0
SAM LAUNCHER	100

BASELINE

VERSION
THREAT CARRIERS
SENSORS
WEAPON CARRIERS
C² ELEMENTS
C² FUNCTIONS
C² CHARACTERISTICS
EVALUATION

C² ELEMENTS

ADOC	1
ROCC	1
SOCC	6
SFC	0
SAMOC	0

C² FUNCTIONS

SENSOR CONTROL
TRACKING
THREAT EVALUATION
THREAT ASSESSED
WEAPON ASSIGNMENT
WEAPON CONTROL
TELLING
EXECUTIVE

TO NEXT PAGE

THREATS CHARACTERISTICS
WEAPONS CHARACTERISTICS
SENSOR CHARACTERISTICS

C² CHARACTERISTICS

DEGRADED OPERATIONS	S
MOBILE	N
NBC HARDENED	N
PHYSICAL SECURITY	Y
SURVIVABLE COMMUNICATIONS	S
BACK-UP CAPABILITY	S
SECURE COMMUNICATIONS	Y
SECURE DATA PROCESSING	Y

EVALUATION

SWaT
CAPACITY
GOAL EVALUATION

SWaT PARAMETERS

	%
# OF RANGE BINS	2
% TRACKED, BIN 1	85
PRIORITY, BOMBERS	1
PRIORITY, CRUISE MISSILES	4
PATRIOT, % CMCs KILLED	85
AMRAAM, % CRUISE MISSILES KILLED	70
% INTERCEPTORS IN BATTLE, BIN 1	20

SWaT BIN 2

SWaT BIN 1

	%
THREATS DETECTED	64
BOMBERS KILLED	95
CMCs KILLED	83
CMs KILLED	34
HAWK REMAINING	0
AMRAAM REMAINING	39

GOAL EVALUATION

	1	2	3	4	5	6
ROBUSTNESS	U	S	P	P	.	.
SURVEILLANCE DATA USE	P	N/A	S	P	P	P
RESPONSE SELECTION	U	P	P	U	P	.
TIMELINESS	S	S	U	.	.	.
AUTHORITY CONTROL	P	N/S	S	.	.	.
SURVIVABILITY	U	P	.	.	.	.
DEGREE OF MAN. IMP/CONT	U	N/S	.	.	.	.
SECURITY/FAULT TOLERNACE	S	N/A	S	P	.	.

CAPACITY

FUNCTION
PROCESSING

PROCESSING

	%
DESIGN LOAD	55
SPARE REQUIREMENT	25

FUNCTION

	%
SENSOR CONTROL	7
TRACKING	19
THREAT EVALUATION	8
THREAT ASSESS/ID	8
WEAPON ASSIGNMENT	26
WEAPON CONTROL	12
TELLING	5
EXECUTIVE	15

89-08-681a

Figure 4-14. CMLP Demonstration Model Concept: Air Defense System

SENSOR CONTROL	
BLANKING CONTROL	M
RADIATION MANAGEMENT	N/D
ECCM CONTROL	N/D
SENSOR TASKING	N/D

TRACKING	
SENSOR DATA ACCEPTANCE	A
COORDINATE TRANSFORMATION	A
REGISTRATION	A&M
FUSION/CORRELATION	A
TRACK UPDATE	A
AMBIGUITY RESOLUTION	N/D
TRACK INITIATION	M
KILL ASSESSMENT	M

THREAT EVALUATION	
RAID COMPOSITION	N/D
FLIGHT CHARACTERISTICS	M
STRENGTH	A&M
PRIORITY RANKING	N/D
ENEMY ORDER OF BATTLE MAINTENANCE	M

THREAT ASSESSMENT/IDENTIFICATION	
FLIGHT ROUTE CORRELATION	A
ROUTE DEVIATION ALERT	A
IFF/SIF PROCESSING	A
GEOGRAPHIC DETERMINATION	M
CHALLENGE PROCESSING	M
DISCRIMINATION	N/D

WEAPON ASSIGNMENT	
WEAPON STATUS	A&M
WEAPON PROBABILITY OF KILL	N/D
WEAPON INTERCEPT TIME	N/D
WEAPON-TARGET ASSIGNMENT	M

WEAPON CONTROL	
INTERCEPT SOLUTION GENERATION	A
INTERCEPTOR GUIDANCE	M
SAM GUIDANCE	N/D
TARGET UPDATE PROCESSING	A
WEAPON EFFECTIVENESS	M

TELLING	
DATA RECEIPT	A
POSITION TRANSLATION	A
DATA TRANSMISSION	A
INPUT/OUTPUT FILTERING (GEO, TYPE, ID)	A
REPORTING RESPONSIBILITY	A
LINK STATUS REPORTING	A
ALERTS/WARNING INFO	A
AUTHORITY CONTROL ARBITRATION	N/D

EXECUTIVE (OS)	
REAL TIME CONTROL	A
SYSTEM MONITORING/RECOVERY*	A
RECORDING	A
SIMULATION	A
OPERATOR INPUT PROCESSING	A
DISPLAY GENERATION	A

EXPANSION OF C^2 FUNCTIONS

89-11-051

Figure 4-14. CMLP Demonstration Model Concept: Air Defense System (continued)

adopted. It had to take into account the fact that the knowledge representation and inference facilities would have to be built, as well as the user interface. The plan required developing the knowledge engineering necessary to support updates and evaluations with simple knowledge represented directly in Prolog, and developing the user interface for the demonstration as an inherent part of the knowledge base maintenance. The knowledge representation stabilized in the initial demonstration model should be converted into an object-oriented form, providing experience with evolving knowledge representations, including the one used in the development of the demonstration system and a second frame-based design study. Section 5.1.4 discusses object-oriented approaches based on Quintus ProTALK and metaProlog.

The TLCSCs implemented and supported by this knowledge engineering effort are shown in Figure 4-15 and described below.
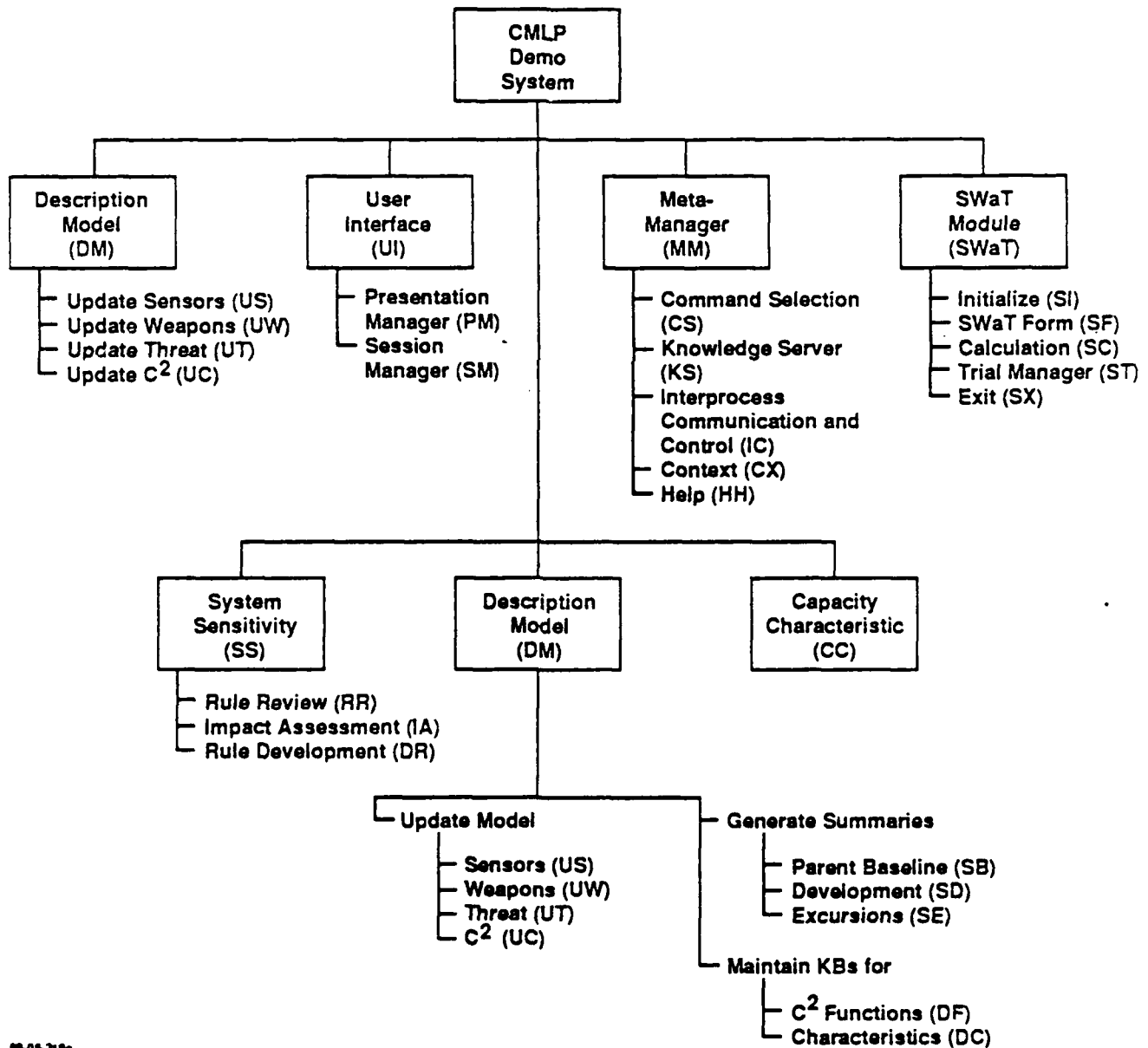
Figure 4-15. CMLP Demonstration TLCSCs and LLCSCs

The class structure and knowledge represen-
tation are described in Section 4.2.2.

- **User Interface.** UI receives commands
from the user and formats responses from the
analytic modules. There are two LLCSCs:
Presentation Manager (PM) and Session
Manager (SM). Rather than being discrete
modules of code, these logical LLCSCs are
implemented as functions of the Quintus
ProWINDOWS package that are distributed
in the LLCSCs supporting each of the models.

- **Meta-Manager.** MM passes relevant facts
to the analytic functions and receives back the
results of their inferencing. MM decides
which function to call, what to pass it, and
what to do with results on the basis of the user
commands passed from SM and the contents
of the context knowledge base. This
knowledge base tracks the inferencing
process; its contents represent the current
state of the system. MM has five LLCSCs:
Command Selection (CS), Knowledge Server
(KS), Interprocess Communication and Con-
trol (IC), Context (CX), and Help (HH).

- **Description Model.** DM maintains the
repertoire of components which may enter the
development baseline. It allows the user to
select from them as appropriate and supply
further details as required to specify a
development baseline, to display summaries,
and to print the current state of the
development system. DM includes nine
LLCSCs which:

(1) perform development baseline system
knowledge base updates:
  - Update Sensors Submodel (US)
  - Update Weapons Submodel (UW)
  - Update Threat Submodel (UT)
  - Update C^2 Submodel (UC)

(2) provide overview summaries of:
  - The "parent" baseline used as a point of
departure for the current development system
(SB)
  - The current state of the development
baseline (SD)
  - The excursions which the user has
applied to the "parent" baseline to create the
current state of the development baseline (SE)

(3) provide for the maintenance and
development of knowledge bases for:
  - C^2 Functions (DF)
  - Characteristics (DC)

- **System Sensitivity Module.** SS describes
the relationships among the components of a
C^2 system. It provides data allowing the user
to reason about (1) the effect of a change in
one component (including threat) character-
istic upon other component characteristics;
(2) the rationale for making the component
changes; and (3) other secondary implications
of the change. SS has three LLCSCs: Rule
Review (RR), Impact Assessment (IA), and
Rule Development (DR).

- **Sensors and Weapons Against Threat
Module.** SWaT allows the user to explore, on a
trial-and-error basis, settings for threat
engagability and the resulting effects on
threat disposal. Seeing the results, the user
can then go back and try new settings for the
relevant factors in order to quickly explore
the effects of differing assumptions. It is
intended to be generic and high level, but
nevertheless provide a ballpark estimate
which can then be developed into a full-scale
study, involving detailed techniques such as
simulation. SWaT has five LLCSCs: Initialize
SWaT (SI), SWaT Form (SF), SWaT Calcula-
tion (SC), SWaT Trial Manager (ST), and
SWaT Exit (SX).

- **Goal Attainment Module.** GA describes
how well a system meets each system goal. For
user reference, the goals, excursions, and
system characteristics which influence their
realization can be displayed. The GA function
also allows display of excursions associated
with user-specified improvements in a given
system goal. GA has three LLCSCs: Goal
Review (GR), Goal Evaluation (GE), and Goal
Developer (GD).

- **Capacity Characteristics Module.** CC
describes the capacity characteristics of each
baseline system in terms of the percentage of
the processing capability necessary to support
various system functions under a specific
threat condition. In addition, percentages
allowed for system overhead, constant

background processing, security overhead, and excess capacity reserve may be specified. The changes in capacity characteristics resulting from an excursion are calculated on the basis of these baseline descriptions.

4.2.2 USE OF KNOWLEDGE ENGINEERING AND INFERENCING.
Many of the possibilities for knowledge representation and inference were not required for the C^2 models listed above. The following comments apply to these features.

4.2.2.1 Objects. Objects have a class-attribute-value structure. Owing to the structure of the model, there is only one well-motivated opportunity to use inference within a class hierarchy, i.e., when both threat carrier and threats can be targets for the SWaT model. Accordingly, it was decided to leave the implementation of the class hierarchy until future development, when an object-oriented system such as ProTALK would be used. The same consideration applied to the provision of multiple facets. However, many attributes were instances of other classes, resulting in a rich structure capable of expansion. Class structure is dictated by descriptive aspects of the conceptual model, the points at which the user would need to change the model, and processing requirements of local computations, such as for the SWaT model.

The class structure is shown in Figure 4-16. There are about 50 classes in the delivered CMLP demonstration system. There are about 500 instances of these classes populating the knowledge bases. The knowledge representations have led to some very complex procedures to access knowledge. These are discussed in Section 5.1.4.

4.2.2.2 Inference. Both data-directed inferencing and goal-directed inferencing are available in LLCSC IA of TLCSC SS. The user may hypothesize excursions and derive one or all impacts (data directed), or he may hypothesize impacts and derive one or all excursions (goal directed).

Inference supported by arithmetic functions is used in the preparation of summaries (DM TLCSC) and capacity and SWaT calculations.

Some inferencing is very difficult owing to the simple data structures. To take a simple case, the browsers often give the name of something and the number of it. But given, say, an instance of Threat Count, Name is not an attribute (see Figure 4-19). To find its name, first the Threat Type must be found and then the Name of that. To find the name of its Threat Carrier is even more complicated.

This seemingly circuitous line of reasoning is due to the requirement of only mentioning the text of a Name once in the system, because only that way can the requirement of complete system modifiability be met. If there is only one possible location for any data item (including the character strings that name things), it can reliably be altered. Concurrent consistency maintenance can then be achieved by updating the browsers that may be displaying that name (see Section 4.3.2).

4.3 USER INTERFACE DEVELOPMENT
The original CMLP design concept was to use a line-by-line interface that would allow the CMLP tool to be portable from system to system. As the design evolved to a set of loosely coupled models, each of which contained interrelated data, it was decided that the line-by-line interface would be burdensome, forcing the user to concentrate on tool use instead of analysis. Accordingly, the CMLP contract was modified to include a windowing package. This package makes the contents of the knowledge base available to the user in a series of forms. He can quickly modify these forms and then determine the impact of his modifications. He is expected in a typical exercise to have many of these forms on display at one time. A very common exercise will be to make a change using one form while retaining several other forms supporting the analysis.

4.3.1 REQUIREMENTS FOR THE USER INTERFACE.
The various models in the CMLP system will require a significant number of forms. For the demonstration system 18 forms were created. Each of these

```
Armament Type              |...Level                      |Goal Status                    |..Range Bin...
|..Name                    |...Number                     |..Name                         |..Tactic...
|..Description             |...Function Status            KB                              SWaT Target Data
|..Target PK               |.....Function                 |..Type                          |..Trial
|....Target                |......Function Group          |..Stored Classes                |..Range Bin...
|....Probability of Kill   |........Name                  |..Required KBs                  |..Type
Baseline                   |........Description           KB Version                      |..Count
|..Version                 |........Function...           |..Type                          |..Priority
|....Author                |........Function...           |..Number                        |..Percent Engageable
|....Date                  |........Name                  |..Author                        |..Initial Number
|....System Type           |......Description             |..Date                          |..Number Engageable
|......Name                |......Func Stat               |..Comment                       |..Killed
|....Number                |......Name                    |..Required KBs                  |..Remaining
|....Parent System Type    |..Characteristic Status       Parent Delta                    |..Percent Tracked
|....Parent Number         |....Characteristic            |..Group                         SWaT Weapon Platform Data
|....Comment               |......Element                 |..Factor                        |..Trial
|....Notes                 |........Name                  |..Old                           |..Range Bin...
|..Delta                   |........Description           |..New                           |..Type
|..Sensor Count            |......Name                    Rule                            |..Number
|....Type                  |......Description             |..Excursion Element             |..Priority
|....Range Bin             |....Char Status               |..Excursion Characteristic      |..Percent in Battle
|......Name                |......Name                    |..Excursion Change              |..SWaT Armament Data...
|....Number                |..Excursion                   |..Impact Element                Sensor Type
|..Weapon Platform Count   |....Element...                |..Impact Characteristic         |..Name
|....Type                  |....Aspect                    |..Impact Change                 |..Description
|....Range Bin...          |......Original Value          |..Rationale                     System PT
|....Number                |......New Value               |..Notes                         |..Type
|..Armament Count          |..Goal Evaluation             |..Applicable Goals              |..Range Bin...
|....Type                  |....Goal                      |SWaT Range Armament Data        |..Percent Tracked
|....Priority              |......Name                    |..Type                          Threat Carrier Type
|....Complement            |......Description             |..Range Bin...                  |..Name
|....Swat Armament Data    |......Goal Group              |..Priority                      |..Description
|......Trial               |........Name                  |SWaT Range Tactic Data          Threat Type
|......Range Bin...        |........Description           |..Range Bin...                  |..Name
|......Type                |........Satisfactory          |..Tactic                        |..Description
|......Priority            |........Partially Satisfactory|....Name                        Weapon Platform Type
|......Number              |........Unsatisfactory        |....Description                 |..Name
|......Used                |........Not Supported         |SWaT Range Target Data          |..Description
|......Remaining           |........Not Applicable        |..Type                          kb
|..Threat Carrier Count    |........Applicable Functions  |..Range Bin...                  |..Type
|....Type                  |......Affects Swat            |..Priority                      |..KBs
|....Number                |......Affects Capacity        |..Percent Engageable
|....Deployment Tactic     |......Affects Geography       |SWaT Range Weapon Data
|....Deployment Range      |......Additional Remarks      |..Type
|......Threat Count        |....Evaluation                |..Range Bin...
|........Type              |....Capacity Evaluation       |..Priority
|........Number            |..SWaT Evaluation             |..Percent in Battle
|....C2 Element Count      C2 Element Type                |..SWaT Ranges
|....Type                  |..Name                        |..Range Bins
                           |..Description                 |SWaT Tactic Data
                           Change                         |..Trial
                           |..Name
```

Figure 4-16. Class Structure in CMLP Demonstration System

forms provides multiple windows. The windows display information, allow the user to make a selection, or provide the opportunity for the user to enter new data. Each form must include a Help facility to explain the nature of the form to the user. The form must include a control capability for turning on Help, and include provisions to save the data on the form, cancel the form without saving the data, provide information to allow selection from other knowledge bases within the system that feed the same form, or allow the contents of this form to be saved as a new knowledge base.

Each of the forms is tiled into separate windows, each containing distinct but related information associated with the form. For example, the baseline display must include a window with a description of the baseline, including author, date created, particular type of system it describes, a unique number, and information about the parent system that may have been used to create this baseline. In addition, the user is provided with a window to enter any comments and a second window to enter any notes to describe this particular system. Windows in the form describe the C^2 elements, sensors, weapons, and threats appropriate for this system. Another window summarizes the goal evaluation model ratings that have been established for the C^2 system. Future CMLP designs may require summary outputs from other models. The final window required in this form is a description of the candidate changes (or excursions) to the baseline that the user is considering.

Since most of the information contained in the baseline display is not controlled from this form, the user will almost always want other forms available on the screen with the baseline display.

4.3.2 THE PROWINDOWS INTERFACE.

The ProWINDOWS package from Quintus was selected to implement the window interface. This package was chosen despite the fact that it was then available only in beta form. It has since been released as a product with an extremely large number of bug fixes.

ProWINDOWS was selected to allow construction of the user interface without leaving the logic programming paradigm. It is a de facto industry standard available to support both Quintus Prolog and BIM Prolog. Other windowing techniques available on the Sun would have required the interface to be separated from the logic programming code, which would have diluted the research into the application of logic programming.

4.3.3 PROWINDOWS DESCRIPTION.

ProWINDOWS (Quintus, 1988b) provides a means for building window-based user interfaces in Prolog. The only commercially available windowing package available for Prolog, it was built by Anjo Anjewierden, a graduate student at the University of Amsterdam, and licensed to Quintus, BIM, and other companies.

ProWINDOWS is an object-oriented system with a class hierarchy and message passing. It provides a number of windowing primitives, including *frame*, defined as a rectangular area of the screen into which many windows may be tiled, apparently limited only by the file descriptor limit in the operating system. Specifically:

(1) A *window* is a rectangular area within which interaction may occur.

(2) A *dialog box is* a window containing structured interaction objects from the following list.

    (a) *Button*

    (b) *Label*

    (c) *Text item*

    (d) *Slider*

    (e) *Menu;* each menu is one of the following types:

      • *Choice* lists all choices and shows selected item as inverted.

      • *Cycle* lists only the current choice and displays all choices when the button is pressed.

      • *Toggle* lists all choices and makes them simultaneously selectable

      • *Marked* is the same as Choice, but check marks, rather than inverts, are used to denote the selected choice.

(3) A *graphical window* contains graphical items including bitmap, box, circle, ellipse, line, path, text, and textblock. This is not used in CMLP.

(4) A *view* is a window containing text.

(5) A *dictionary* is a structure maintaining a mapping.

(6) A *browser* is a view consisting of lines of scrolled text.

4.3.4 PROWINDOWS PROBLEMS AND SOLUTIONS.

We found a large number of problems in the ProWINDOWS implementation. To overcome them we had to make substantial changes in the overall CMLP architecture and also provide a large amount of code that runs above the ProWINDOWS code to allow the ProWINDOWS package to be useful.

Many of the difficulties discussed here will be resolved by PL/X, a completely redesigned user interface that will run under Xwindows (rather than SunView). This may provide a much more portable and powerful user interface that can still be programmed in Prolog.

4.3.4.1 File Descriptor Problem.

A major limitation was the ProWINDOWS reliance on SunView in the Sun operating system. Each of the windowing systems available from Sun uses file descriptors for windows. In Sun OS Version 3.X, there is a limit of 32 file descriptors per process. This limit is increased to 64 in Sun OS Version 4.X. Each SunView window (which corresponds to a CMLP box) requires at least two file descriptors. A complex SunView frame (or CMLP form) can easily come close to the 32-file descriptor limit when reserves are left for popup windows callable from the form.

We identified three solutions to the file descriptor problem. The first was to limit the use of windows to allow implementation with 32 file descriptors. With the current form designs, this would typically have limited us to only one form at a time. Redesigning the form to use fewer windows appeared to be impractical. Further, the one-form limit would have negated many of the advantages of

a windowing system. The second solution was to use Sun 4.0. This was not available to us during the critical portion of the development. If it had been, it would still have allowed only 2 to 3 forms within the file descriptor limitation of 64. The third solution, and the one selected, was to run the CMLP system as a multiprocess rather than a single process. This approach created two secondary problems: creating an architecture for the multiprocess system, and accomplishing interprocess communications within the ProWINDOWS limitations.

We considered two alternative multiprocess architectures: chain (rejected) and star (selected). The chain architecture starts with a root parent origin process. Each subsequent process except the last has a parent process and a child process. Parent processes communicate with children and further descendants by sending messages through the standard Quintus IPC mechanism. Each child examines the message for its address and either processes the message contents or passes a message on to a subsequent child. Children can communicate with parents by returning results from a parental message using the standard Quintus IPC mechanism. Children are not able to initiate communication without modification to the IPC mechanism. In this architecture, new processors need to be spawned only when the current child process runs out of file descriptors for its window operations.

One problem with the chain architecture is that considerable system overhead may be spent and time delays may occur as information is routed through several processes before reaching its destination. Also, once a process has been placed in the chain it cannot be removed without breaking the chain. In a complex CMLP operation this may cause many processes to have to remain in existence to maintain the chain, considerably limiting the available system functionality.

In the star architecture there is one central process or server. The server can spawn any

number of child processes. Each process is responsible for a different system function (interface form). Each child process can spawn a child of its own to handle functions such as a popup window. This architecture overcomes the problems with the chain because a process is only active as long as an interface window is being used. As soon as the user selects DONE from the interface menu, the window and the process are destroyed. Additionally, message paths are shorter for the star architecture because most messages will be between the server and its child (or from one child to a subsequent child).

One problem with the star architecture is that the developer has to specify the various child processes that may be created. Each child process is geared in this specification for a specific function. This limits development flexibility and requires modifications to the server to expand the system. On the other hand, it aids in structuring the system.

Both chain and star architectures can create communication and timing sequencing problems. If the user is trying to do several operations at once, communications may get out of sync. Also, the polling process required for child-to-parent communication adds to overhead.

To implement the star architecture it was necessary to extend the IPC mechanism provided by Quintus. The existing IPC mechanism would have limited us to a master Prolog process and a servant Prolog process with only one-way control. The master is allowed to send requests to the servants and may receive responses when the servant has completed the task. There is no mechanism for the servant to send a request to the master. This mechanism is appropriate for controlling popup windows when the system needs some input from the user. It does not work well for more complex forms. The one-servant limitation would have required disconnecting a servant process and reconnecting a new process to use a new form.

However, different servant processes could handle different tasks working only one at a time. We modified the Quintus IPC mechanism to allow multiple children from one parent and to allow children to initiate communication with the parent. The server must sequentially poll the communication links from its children and process any messages that come in.

An alternative is to run isolated ProWINDOWS processes. This would have alleviated the file descriptor problem but would have caused major integration problems.

We also considered using a library routine, Popen, available within Quintus Prolog. Popen allows one Prolog process to start another Prolog process and communicate with it. The communication is one way, reading from standard inputs or writing to standard outputs. While this allows multiple Prolog processes to be hooked up, the communication limitations appeared unacceptable.

In summary, the selected approach of the star architecture required modifying the IPC mechanism. The first modification allows the master process or server to control any number of child processes. Each child process may control its own popup windows or other subchild processes. A second modification allows two-way communication between server and child. To implement two-way communication, the server must poll each active child periodically to establish a request from the child for information transfer. However, this solution involves creating separate users for each form, from the perspective of the Sun OS. It thus requires the interchange of information between forms to occur through the OS and, as stated earlier, lost the rapid-prototyping benefits of Prolog.

4.3.4.2 Low-Level Facilities. We found that the structures provided by ProWINDOWS are in most cases of too low a level to be used directly. A number of the limitations of ProWINDOWS seem directly related to its use of the SunView user interface capabilities. A similar user interface is HyperCard (Goodman, 1988). HyperCard uses a subset of

the Macintosh user interface tool box. This tool box (Apple Computer, 1985) provides much higher level structures than does SunView (Sun, 1986). To alleviate these problems we created CMLP-unique forms, browsers, and composite browsers, as well as several processes to handle messages and data within the system. These CMLP-unique facilities are discussed in Section 4.3.5.

4.3.4.3 One Selection Per Screen. Sun-View limits the user to only one active selection per frame. This means that as the selection is made, highlighting is dropped for other selections in the system. To ameliorate this problem, a small diamond is available to mark a particular area. This small diamond is inserted by SunView underneath and to the left of the selected item and is not very noticeable or clear as to which element has been selected. An example of this problem is shown in the system sensitivity form. ProWINDOWS provides no facilities to alleviate this problem.

4.3.4.4 Message Limitations. Within ProWINDOWS, certain messages can be sent · to windows only before they are open (i.e., displayed) and other messages can be sent to windows only after they are open. This division is not apparent from the documentation and is derivable only from an understanding of the ProWINDOWS internal structure not obtainable by reading the manual. These limitations cost extensive time in attempting to use ProWINDOWS.

4.3.4.5 Menu Limitations. Menus are fixed at the time of their creation and cannot be extended. Where we expected data entry to create new elements in a menu, we had to implement an awkward form for that menu. For example, the range bin box in the sensor/range form uses a browser to list the number of bins. Bins can be specified only for using the sensor range form. Other forms use a cycle menu, with the number of range bins defined by the sensor/range form. Similarly, on the slot model form the child is selected from a browser to add new tiles and delete old ones. Menus contained enough built-in

structures within ProWINDOWS to support display and selection, while browsers required considerable support.

4.3.4.6 Browser Limitations. Browsers and other windows in the CMLP system were limited by an inability to define a name stripe. A name stripe is an inverted title above each form and is available for a frame but not for each individual window. To overcome this limitation, we provided a name stripe identifying columns within the browser field as a lead item. Unfortunately these titles are not fixed on the screen and scroll out of the user's view as he reviews elements of the browser. If he needs definition of the columns within the browser, he must go back to the beginning.

ProWINDOWS did not provide any support for marking the beginning and end of a browser field. We overcame that limitation by providing a dotted line at the beginning and end of the data within a browser. The beginning stripe falls immediately underneath the title after each column discussed in the preceding paragraph.

Most of our browsers use buttons to control data entry associated with the browser. This caused us to create a composite browser as discussed in Section 4.3.5.6.

4.3.4.7 Selection Limitations. We created several browsers that are used as a menu to let the user make selections. Unfortunately the user cannot select elements that are stored in columns within one browser item. This required us to use a popup window to support changes to elements within a browser line.

4.3.4.8 Data Justification. When gaining initial experience with ProWINDOWS, we decided to build a simple spreadsheet form to investigate some of the structures in the SWaT and capacity models. We found that text items can only be left-justified, which meant that there was no way to present numbers so they could be edited in a natural way; at best the item would have to be reformatted and redisplayed after editing. (Quintus told us that one of their programmers had attempted to build a spreadsheet

with ProWINDOWS and failed for precisely the same reason.) We therefore had to provide a process to right-justify numbers after entry. This process also prevents knowledge base items from being simply truncated when they are entered into a field of limited size.

4.3.5 CMLP INTERFACE STRUCTURE. In addition to the data justification process described above, we created the following seven special processes to run above ProWINDOWS

4.3.5.1 Form. A form is a collection of rectangular regions called "boxes" that are fitted into a rectangular space by the Quirtus ProWINDOWS software. The form is the user interface for each user-selectable component of the CMLP prototype system. There is exactly one form per saved state of the interface processes.

Each form is divided into rectangular boxes having different responses to mouse clicks within their boundaries. They also have differing types of display format. These aspects are customized to the subfunction they perform. Similarly, as Figure 4-17 shows for the sensor/range development form, selection or update of a box leads to other boxes being updated, as indicated by the arrows.

Each form is created, used, and destroyed in a single clause of procedure review or evaluate. This clause can extend over several printed pages. Clauses of such size are not usually considered good Prolog style but are required by the large number of variables bound when making components of the form that need to be accessed by other components. The problem is exacerbated by some ProWINDOWS classes (such as browser) that can only be initialized after the frame has been opened. The convention used as a model for these clauses is:

(1) Create frame with form title.
(2) Create standard application bar.
(3) Create boxes.
(4) Specify layout relationships.
(5) Open frame (i.e., the form appears on the display).

(6) Initialize boxes.
(7) Service user requests.
(8) Destroy frame (i.e., the form disappears from the display).
(9) Report updates to meta-manger as appropriate.

4.3.5.2 Standard Application Buttons. Below the title bar of every form is a box containing one or more of the standard application buttons. These operate on the entire form, and their effects are identical across forms. All buttons remain inverted while their associated code is executing. The effects of clicking on a button are as follows.

● *HELP.* If the Help system is off for this form, it is turned on. Subsequently, a Help text is displayed in the Help form when the user causes the mouse to enter a box. The Help text resides in a file with the same name as the box name, and may be updated with an

Title Bar
Standard Application Buttons
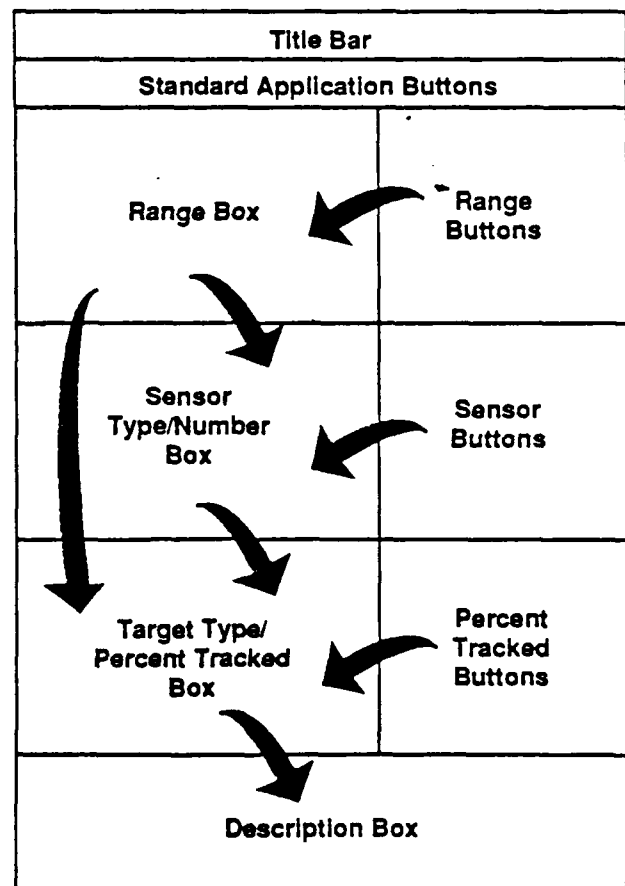
Figure 4-17. Schematic Diagram of Sensor/Range Development Form

external editor. Upon detecting the message for mouse entry to a box, the current text in the Help form is replaced with the text in the file. If the Help form is not displayed, it is invoked.

If the Help system is already on for the form when the button is clicked, it is turned off, ignoring subsequent entry messages. If no other form has Help turned on, the Help form is destroyed.

- **DONE.** Updates (if any) made since the form was invoked are reported to MM and the form is destroyed.

- **CANCEL.** The form is destroyed without any updates being sent to MM.

- **LOAD.** The user is presented with a popup window listing the indexed saved versions of the knowledge base displayable by the form. The popup window is a composite browser with summary data about each version in the browser, and with SELECT and CANCEL as application buttons. If the user clicks SELECT the most recent selection in the browser is loaded, replacing any version already loaded. If there is no selection within the browser, clicking SELECT has no effect. If the user clicks CANCEL instead, the LOAD is canceled; any previous version remains in place. It is not possible to abort a load once it has started.

- **SAVE.** The user may be prompted for descriptive information about the version being saved in a popup window. The version is indexed by the system and the relevant instances are written to disk. The system retains data allowing access to the stored version. It is not possible to abort a save once it has started.

All buttons are made by the procedure make_buttons/3, in which each button is specified generically as:

   Name-(Head:-Body)/ButtonID

where

Name is the name of the button, such as HELP, above.

(Head:-Body) is the procedure called when the button is pressed. The procedure makebuttons asserts (Head:-Body) and sends

a message to the button; hence this procedure matches the message cascaded from the user interface when the button is pressed. Such an approach allows variables to be bound within the form as required for the specifics of the button semantics. Binding variables in this way obviates a considerable amount of context data saving and referencing.

ButtonID is an atom tagging the button's reference in the context knowledge base. It is only needed when a button is created that can present messages from the knowledge base.

4.3.5.3 Generic Application Buttons. The generic application buttons and application-specific buttons (Section 4.3.5.4) usually form part of a composite browser and are used to control the effects of selection within the browser. The generic application buttons have the same meaning across all browsers to which they apply, while the application-specific buttons have specialized meanings, specific to the form they are in and the browser to which they are attached. The generic application buttons are as follows:

- **MODIFY.** This button allows the user to modify the values in one or more columns of the current selection of the associated browser. A popup window appears, with a dialog box showing details of the current browser selection. The user may modify some of those details, by typing a new value for the named attribute. The application buttons attached to the dialog box are ADD, DONE, and CANCEL. The ADD button is described below. If the user clicks on DONE, the updated values that the user can legally edit are used to update the local knowledge base and the appropriate browsers in the form are reinitialized to display the effects.

- **ADD.** This button allows the user to add an item to the associated browser. It pops up a composite browser initialized to a list of all instances of the class displayed by the browsers that are not currently in it. The associated buttons are NEW, DONE, and CANCEL. The NEW button is described below. If the user clicks on the DONE button, an entry is appended to the calling browser in

which the attributes are given minimal values (e.g., 0) and the ADD popup is destroyed. If the user clicks on the CANCEL button, no update is made and the ADD popup is destroyed.

- **DELETE.** This button deletes the current selection from the associated browser and updates the local knowledge base as appropriate.

- **NEW.** This button pops up a dialog box with attribute names and text items which allow a new instance of the associated browser's class to be specified.

- **STORE.** This button (together with its dual, REVERT) allows the user some capability to undo excursions he has made. Pressing STORE preserves a state that can be REVERTed to.

- **REVERT.** This button allows the user to undo all the changes he has made within the form since he last pressed STORE (or the form was initialized).

4.3.5.4 Application-Specific Application Buttons. These buttons provide specialized actions specific to the form. They are made by make_buttons/3, described above. Some examples follow.

- **NEW (Range).** Allows the user add a new range bin to the development baseline.

- **NEW (Trial).** Supplies the user with a new trial in the SWaT model.

- **CALC.** Performs the SWaT calculation and displays the results.

- **SELECT.** Provides the same functions as above, but within the browser rather than a form.

- **DESELECT.** Removes all selections from the rule excursions or impacts, so that the user may specify the conditions for a FIND. The user selects excursion and impact elements, characteristics, and changes as required.

- **FIND.** Finds the first subsystem sensitivity rule matching the conditions selected. If the button remains inverted after the matching rule has been displayed, there is another matching rule. Another click on FIND displays it.

4.3.5.5 Browser. The ProWINDOWS class browser is not usable for menu selection without considerable supplementation. Not only does it lack fundamental structures necessary to support all but the simplest choice sets, but it also lacks programmer conveniences. For example, the width of a browser is the width of its frame, not the width of its text display area. Browser width thus cannot be calculated from the width of the strings; a vertical scrollbar, if any, and the pixels at the edge of the frame must also be taken into account.

The CMLP browser is illustrated in Figure 4-18, which shows initialization processes. The ProWINDOWS browser (at the top of the figure) is supported by a ProWINDOWS dictionary (as recommended in the ProWINDOWS manual) to hold the constants displayed in each successive line. But the dictionary entries must be formatted, and must include header and footer lines as described above. Entries are truncated to allow them to fit within a column. The formatting is controlled by a display_format/2 specification, which specifies the descriptive text at the top of each column, the column width in the browser, and characteristics to be used when an expanded version is to be used in a popup for update.

The input to the formatting process is not the instances themselves, but the output of init_values/3. This procedure may need to access instances of multiple classes from the knowledge base and may need to prepare summaries and other abstractions of the data. init_values/3 produces two outputs. One is a list of lists of the values (dict_items) to be displayed, one item per column, and the other is a corresponding list of the instances so presented. These lists are amalgamated with the list of formatted entries so that when the user makes a selection (returning a formatted entry), the instance or list of values can be identified as required.

The context knowledge base is used to access the browser and dictionary references
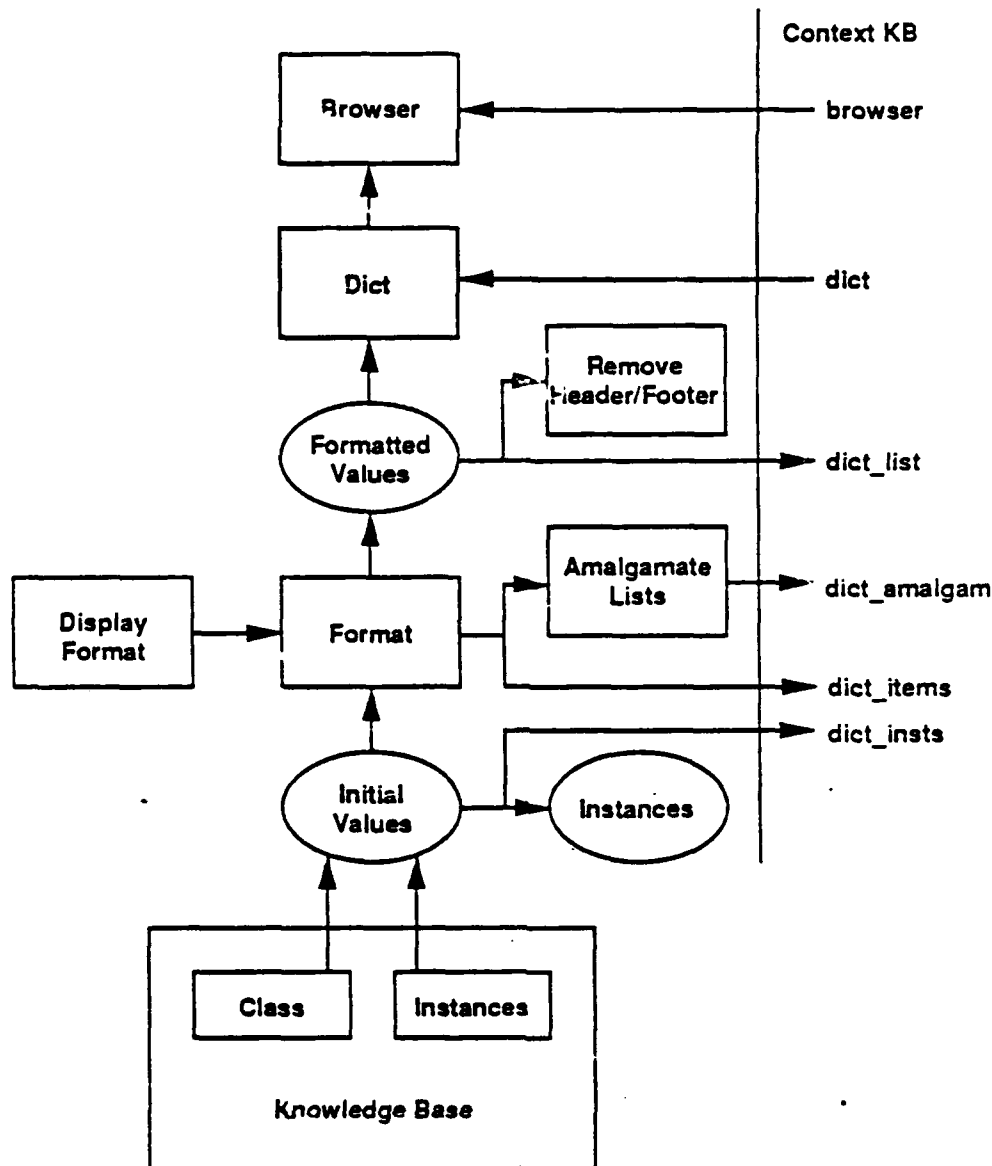
Figure 4-18. Browser Initialization Processes

(which are previously created and stored in the context knowledge base by the call to make_browser/[3,4,5]) and store the instances, items (list of values), and amalgamated list, and to set the current selection to the empty string.

4.3.5.6 Composite Browser. A composite browser is a CMLP browser accompanied by a dialog box containing application-specific application buttons. For user interface uniformity, the dialog box is displayed adjacent and to the right of the browser.

4.3.5.7 CMLP Menus. CMLP menus include the following.

• *Marked Menu.* This is the same as in ProWINDOWS, but may have initialization and other processes associated with it.

• *Toggle Menu.* This is the same as in ProWINDOWS, but may have initialization and other processes associated with it.

• **Switch.** This is the same as the ProWINDOWS cycle, but may have initialization and other processes associated with it.

• **Cycle Menu Bar.** This is a dialog box containing a row of switches.

4.3.6 LESSONS LEARNED AND RECOMMENDATIONS REGARDING THE USER INTERFACE.

The prime focus of the CMLP project has been logic programming. Unfortunately, the use of ProWINDOWS resulted in numerous problems which detracted from the logic programming effort. There are several lessons.

(1) Originally we developed the knowledge base and user interface in isolation, but experience with translating into non-interface form led to some revision of the knowledge base structure. It is not clear that the two components can be developed and tested in isolation.

(2) We should move to a window mechanism which does not depend on Unix file descriptors. One of the significant problems with ProWINDOWS (actually with SunView) results from the use of file descriptors by windows. Windowing systems currently under development, using Xwindows and NeWS, eliminate the dependence on file descriptors and will greatly simplify future user interface integration.

(3) SunView is an extremely powerful application-development system for user interfaces. Unfortunately, ProWINDOWS uses only a subset of SunView, and does so in an inconsistent manner. Careful thought should be given before using ProWINDOWS in future developments. This leads directly to the next point.

(4) One of the primary reasons for using ProWINDOWS was the requirement to investigate the production of conceptual modeling software in Prolog. Unfortunately, not every language or software development system is well suited to every application. Software languages/environments must be carefully matched with application requirements so that time and effort are not wasted. In the case of CMLP, Prolog is ideal for the logic programming necessary for knowledge acquisition and manipulation. But without better tools, Prolog is not suitable for building user interfaces.

The object-oriented nature of ProWINDOWS (which effectively provides a subset of Prolog capabilities, but in a very useful form) provides classes and message-passing and is clearly a step in the right direction. But the implementation of these ideas is incomplete, and (in the version used in this study) too unreliable to properly indicate the power of the approach.

5. EVALUATION OF CMLP RESULTS

This section describes the extent to which the CMLP research project has demonstrated the use of conceptual modeling and logic programming in command and control system design. Section 5.1 focuses on the demonstration system and Section 5.2 on the design of the overall CMLP tool of which the demonstration system forms a limited part. Although the project did not include evaluation of the utility of the models in the demonstration system, it fully substantiated the applicability of the methods and the soundness of the design. The tool is extremely easy to use and greatly benefits the organization of the C^2 design process. Logicon accordingly plans on further evaluation of the tool during the course of C^2 design projects and system architecture activities.

5.1 USE OF CONCEPTUAL MODELING AND LOGIC PROGRAMMING

5.1.1 CONCEPTUAL MODELING IN CMLP

5.1.1.1 Extent of Conceptual Modeling Use.
The demonstration model was designed to create the underlying framework of objects, relationships, and properties that are important in the conceptual modeling domain. However, the delivered demonstration software does not always work to implement the characteristics described below. Briefly summarizing only some of the major components:

- **Description Model.** The user can easily create, modify, or view descriptions of existing or hypothetical C^2 systems.

- **SWaT Model.** After defining particular baseline system descriptions, the user can execute different attack scenarios to see how particular combinations of sensors, weapons, and C^2 rules will perform against the specified threat. The SWaT model, a tool for rapidly exploring alternatives during the early design process, provides estimates that serve to narrow the range of solutions to a particular problem. After the candidate solutions have been found, detailed simulations may be run to determine the final quantitative projections.

- **Capacity Model.** This model provides first approximations of capacity bottlenecks in an existing or hypothetical system.

- **Goal Attainment Model.** This model allows the user to evaluate a C^2 design against a series of performance and survivability goals. He may then use the effect on the goal evaluations to determine the desirability of design excursions.

- **System Sensitivity Model.** This model provides explicit ways to represent and make deductions about particular causal relationships that exist in the domain, so that the user can identify the effect of a change on all elements of the system.

It is clear that the models described above are representations of conceptual models. How they relate to some of the descriptions of conceptual models available in the literature is not so obvious. Section 2.2 describes a potentially applicable framework, the Gaines hierarchy of conceptual models [Gaines, 1988]:

1. *Source System:* Distinctions among fundamental concepts/interactions with the world

2. *Data System:* Representations of objects and relations

3. *Generative System:* Inference based on data systems

4. *Structure:* Comparison of generative models

5. *Meta-System:* Abstract comparison of models

6. *Meta-Meta System:* Reasoning based on abstractions of the models

In many cases the CMLP model as implemented in code corresponds to an element of the hierarchy one level higher that the corresponding process applied by the user and the CMLP model. Each element of the description model is a source system, since these elements relate to distinctions that happen in the real world. In populating the description model or design/requirements traceability model, the user records his perception of the system in descriptive form;

these models are of the nature of data systems. However, as the user makes changes within the description model, he and the CMLP system are performing as a generative system. Computational models such as the cost and SWaT models are data systems. The analogy development model is a generative system; however, the user performs meta-system analysis by comparing the nature of C^2 systems to define whether their similarities are sufficient for analogies to be valid.

We concluded that the user and the models are acting together as a pair, and that the user extends the models in his use. In trying to rank the objectivity of the individual models (Figure 5-1), we saw a tendency to place the model with user interaction as being more

Capacity Model
Cost Model
Requirements Traceability Model
Simulation Interface

SWaT Model

System Sensitivity Model

Goal Evaluation Model

Description Model
Analogy Development Model

Objective

Subjective

90-05-026a

Figure 5-1. The Nature of CMLP Models

subjective than the model by itself. For example, we rated the SWaT model as clearly more subjective than other models because the user must determine the parameters required by SWaT in a highly subjective manner.

To summarize the extent to which CMLP uses conceptual modeling, we believe that the tool draws heavily on the process. It is a significant user aid for C^2 design processes. It does not replace the C^2 design expert.

5.1.1.2 Approach to Development of Conceptual Models. Section 2.4 presents two alternative approaches to developing conceptual models: top-down methodology and exploratory programming. We attempted to follow the latter approach, but with four significant differences.

First, problem identification was not completed until well into formalization. While the methodological model allows for cyclical iteration, the project showed that it is possible to rely too heavily on this flexibility. The ability to go back and make fundamental revisions well into model development may result in a better or more useful tool—if project resources can bear the load that this iteration creates. This is a major concern, given the extent to which we found conceptual model development to be goal driven. Until the goals are finally agreed upon, the model may undergo major revisions.

Second, we had to make major revisions of the conceptualizations because the salient objects in the domain and their relationships, as well as the resolution at which they should be represented, were goal dependent. As the conceptualizations became richer, the potential system users were seen in a different light by the domain experts. This led to revisions in problem formulation. Considerable iterations occurred between these two stages.

Third, the formalization and implementation stages occurred together. Rather than using a preliminary, exploratory prototype for formalization, we implemented the final tool by expanding its scope as formalization proceeded. This was a necessary concession to
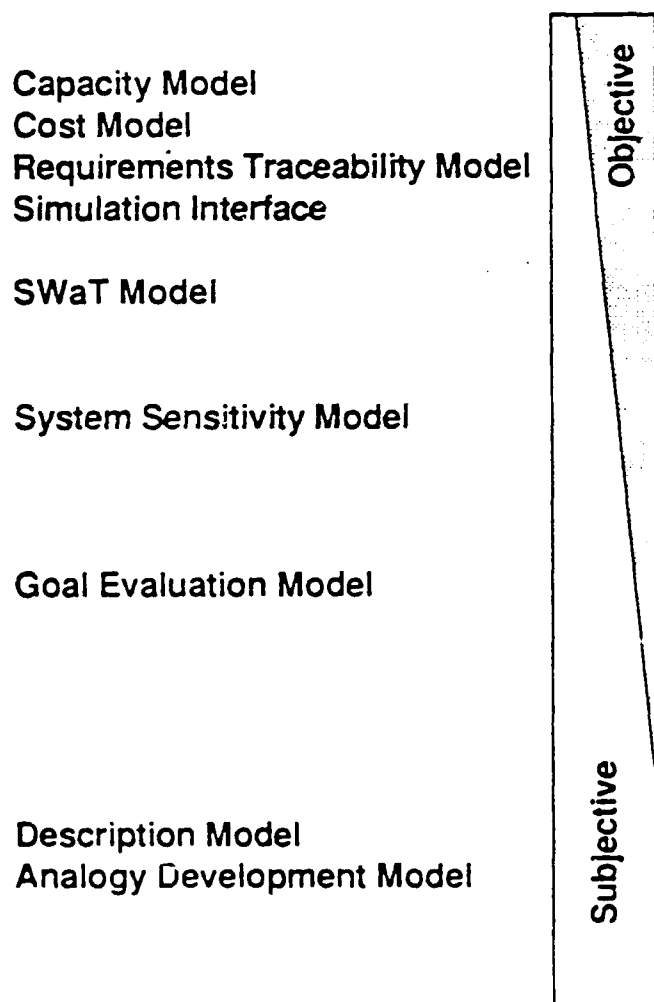
limitations of project resources and schedule. From this aspect of the experience, we conclude that model construction and refinement should be separated from the development of the other software subsystems going into the deliverable system. Different kinds of engineering controls are appropriate for these two processes, and attempting to mix them was unsuccessful. The formalization stage was highly interactive with conceptualization and therefore required maximum flexibility (especially in view of the late revisions of the goals). The implementation of the deliverable system should have been governed by a rigorous engineering discipline, but this would have precluded the necessary flexibility. To a large extent the two requirements are incompatible.

Last, the concept of continuous revision, converging on a final implementation using feedback from experts, was central to the effort. Difficulty arose when fundamental concepts about what could be done with the model changed as the domain experts gained experience with the partially implemented version. The ability to make revisions should not be used to justify incomplete problem specification; it is intended to refine rather than to redefine.

5.1.1.3 Observations From the CMLP Experience. The basic project goal was to study the process of conceptual modeling using logic programming in the arena of C^2 system design. We studied the process by posing a practical problem in conceptual modeling and attempting to solve it using the tools available. We do not claim that the resulting observations which follow are fully generalizable to other conceptual modeling problems, but believe that they shed some light on some underlying problems and constraints associated with the modeling process in this domain, and the implications that these may have for other domains.

The premises that underlay the study are significant because they resemble the conditions under which much ad hoc expert system development takes place in industry and government: a need is perceived within a user community and an effort is launched to meet it. Our study had the following characteristics:

- The problem domain, the C^2 system design process, was initially ill defined.
- The knowledge engineering resources, especially in terms of expert availability, were constrained. There was, for example, no opportunity to sit with a domain expert as he worked through a real system design problem.
- There was no precedent for the target application area.
- The conceptual modeling process was directed at how human system engineers perceive and think about C^2 systems, rather than physical, real-world systems (for the modeling of which there is considerable precedent, although not using logic programming).
- There was significant expert disagreement about both the representation of the domain and the uses to which a resulting computer-based aid could be put.
- The study focused on logic programming as a paradigm for conducting the conceptual modeling of C^2 system design. Normally, this would not be a real-world constraint. It was in our study because of the overriding research interests.

The above characteristics clearly do not constitute the ingredients for a successful top-down software engineering effort. In the context of these specific conditions, we draw the following general conceptual modeling observations from the CMLP experience.

First, the conceptual modeling process is iterative and cyclical, rather than a linear sequence of steps as discussed in the previous section.

Second, expert disagreement is likely to increase when the role of human judgment in the modeling process is emphasized. Experts will disagree not only on the model itself, but on its scope and use. This phenomenon reflects the limitations of the shared conceptual model maintained by experts with different (and different degrees of) experience

in the domain. One of our important observations is that such disagreement is inherent in the situation and cannot be avoided. However, to the extent that the resulting conceptual model highlights these differences, it may be performing an important service. Expert disagreement should be dealt with as an asset, but the expectations of the experts should be conditioned to this view of the situation. We tried to make the model highly adaptable to allow experts to "personalize" it and also tried to make areas of potential difference explicit to allow resolution.

Third, because of individual differences in domain perception, conceptual modeling is highly goal driven. The structure and contents of a model cannot be separated from the problem-based context within which it is constructed.

Fourth, the development of a conceptual model should be distinguished from its implementation in software as a deliverable prototype as discussed in Section 5.1.1.2.

Fifth, the problem of confounding knowledge acquisition (i.e., model construction) and model implementation was exacerbated by the absence of a high-level development environment for logic programming which would have absorbed much of the burden of configuration control, truth maintenance, consistency checking, and so forth, and which would have provided "tool kits" for inferencing, user interface construction, and so forth.

In sum, we found that it is extremely important for the experts being consulted in model development to interact with the emerging model; indeed, it is hard to imagine that we could have converged on a useful solution without such "user-centered" software tool design. We therefore conclude that the use of a development environment is extremely important under conditions like those of our study It is probably essential if the deliverable prototype is to be derived directly from the working prototype. We did not have the advantage of such an environment.

5.1.2 LOGIC PROGRAMMING IN CMLP.

Logic programming using the language Prolog has contributed to a valuable tool for addressing C^2 system design. The CMLP demonstration system incorporates a sophisticated window-based user interface composed of multiple interacting processes in Prolog. It was created in about 10 man-months, including time to learn via trial and error the actual capabilities of ProWINDOWS and the knowledge server engineering. A large portion of the effort was devoted to resolving problems due to the immaturity of the ProWINDOWS package.

One test of the use of logic programming is the extent to which rule-based reasoning is used. The CMLP model, as currently implemented, uses only simple rules of the form:

If (characteristic X) of (object 1) (changes), then (characteristic Y) of (object 2) (changes).

This rule structure is the basis of the system sensitivity model.

A more complex rule structure was coded into the capacity model but does not currently work within the demonstration system. An example of the rule used by the capacity models is:

If a percent of capacity is currently required by function X, function X linearly increases in capacity demand with threat, and threat increases by B percent, then function X will require A + .01 AB percent of the total system capacity.

Analogy development (which is not implemented in the demonstration model) was designed to use:

If change X causes effect M in system A and X is similar to Y and A is similar to B, then change Y is predicted to cause an effect similar to effect M in system B.

These rules can be used to evaluate the impact of change Y in system B or to determine what type of change is needed to system B based upon experience with system A if a particular effect (similar to M) is desired.

Use of the traceability model involves a rule structure as follows:

If function X produces only effect A in system M, and system M no longer requires effect A, then system M no longer requires function X.

The cost model will establish a set of relationships between element in the C^2 system and the cost of the C^2 system. These relationships imply rules for inferring change to C^2 system cost as elements of the C^2 system are changed.

Section 2.6.3 developed a list of advantages of using logic programming. These are evaluated below according to their realization in the demonstration system. Many of the features are so ingrained in logic programming that they are used unconsciously by an experienced logic programmer, for whom they are second nature.

1. General Record Structures. These are used throughout the code. A specific example is class definitions for user interface procedures (see init_class/2 in CMLP code). These are defined as a list of constants. Yet some of items in the corresponding elements (see inst/3 list) will be:

- Integers, as in the number of AMRAAMs on an interceptor
- "String" constants, as in the name of a type of armament (e.g., "AMRAAM")
- Terms, representing instances, as in 'Function'(3)
- Lists of the above

No attention is paid to the differences between these in designing code to set or get data.

The list itself is readily extensible; the procedures that set or get values (set_vval and get_vval) do not expect a certain number of attributes. They simply work uniformly with what they find.

2. Built-In Pattern Matching. Pattern matching is used every time a goal is used to select a procedure; it is used throughout the code. One example is the procedure

make_buttons/3, which has five clauses. The first is

Clause 1: make_buttons(Object,[ ],[ ])

where Object is the reference to the box in which the buttons are placed. This is a recursive procedure. The second argument is a list of button specifications and the third a list of corresponding button items implementing the functionality. Since the second argument is nil (no buttons to make), so is the third; this clause is matched at the bottom of the recursion, with the output list (third argument) being built on backtracking.

Each of the other four clauses in the procedure implements a special case of what is needed from the button. This is picked up by matching the form of the head of the second argument, thus:

Clause 2: Name-(Head:-Body)/ButtonID
Clause 3: Name-(Head:-Body)
Clause 4: Name-Pred
Clause 5: Name

The arithmetic operators provide a simple way of structuring related items, and have a well-defined precedence. This is easier to understand and more efficient than using a Lisp-like list [Name, Head, Body, ButtonID], although this would be equivalent. Each clause picks up a successively less specialized case. In Clause 5, the button only has a name; no action is supplied. In Clause 4, make_buttons supplies the code to invoke procedure Pred when the button showing Name is pressed by the user. The predicate Pred must be asserted by the user or otherwise supplied. In Clause 3 the procedure to be invoked (Head:-Body) is asserted when make_buttons is executed. (It also retracts previous versions.) This allows the programmer to construct the procedure at execution time. In the CMLP code, this was done to minimize the number of calls on the context knowledge base; ProWINDOWS-generated references (addresses) can thus be hardwired into the button-response code. In Clause 2 an

external ButtonID tag is supplied so that access to the data for the button can be obtained from the context knowledge base from external code. This would allow it to invert or gray the button as an indication of some internal state to the user.

3. Multiple Inputs and Outputs. An example is init_values/3, where the first argument is bound and is the internal browser ID, the second returns the list of items to be formatted and displayed, and the third is a list of the instances displayed in the second. The absence of fixed commitments to input and output variables is a useful feature, but is limited by the true reversibility of the procedure. Usually procedures containing arithmetic evaluations or comparisons are not reversible. It is, however, usable for ground terms and is frequently used in debugging when the programmer wants to ask such questions as "Where is this data item stored?" or "What data item is stored here?"

4. Both Declarative and Procedural Readings. This feature is inherent in CMLP, as in any Prolog program.

5. Intensional and Extensional Data. This feature was used often used during program development.

6. Separation of Logic From Control. This feature is inherent in CMLP, as in any Prolog program.

7. Nondeterminism. This feature was not truly used in the demonstration system.

8. The Logical Variable. This is used throughout CMLP. See the discussion under items 1 and 2 above.

9. Natural Interface to Database Managers. All of the active CMLP data are present in core. An extended dataset for storage and reference through the analogy development model to knowledge bases describing other baselines could exploit this.

10. Conceptual Level of Representation. This was the primary motivation for using logic programming for this task. Although some structuring is imposed above "vanilla" Prolog, very little is needed (see the comments under item 1 above).

Logic programming enabled us to achieve a very complex program in a very short time, despite the fact that the tool set we used (Quintus Prolog and ProWINDOWS) was immature and provided very little in the way of development facilities. However, we note that the Quintus offerings are easily as mature as other Prolog tools (such as available from BIM). The next section summarizes recommendations for development environment.

5.1.3 DEVELOPMENT ENVIRONMENT NEEDS. A more powerful development environment for Prolog programs than that of a simple debugger-tracer is long overdue. Although Quintus supplies an emacs-based program development interface that "knows" where procedures and procedure boundaries are located in the source code, this facility does little to support programmer productivity. Extensions needed include:

• Browser-mediated access to data within the existing code for indexed access to procedures and list of local variables

• Aids to identify instantiation structure, including procedure calls (e.g., when the code requires a certain number of arguments or a certain range of structures, this fact should be identified to the programmer)

• Intelligent program development procedures which would look at the code and indicate, for example, that the procedure call had three arguments when the programmer was changing the definition to four

• Intelligent assistance covering common coding mistakes

• Test procedure library maintenance and background invocation

• Graphical displays of program structure

It is an extremely poor use of cycles on a single-user workstation to wait for the user to "try out" a change (usually involving a real-time wait for recompilation and possibly screening through copious quantities of tracing output) when the declarative properties of logic programming and the ready availability of inference make background checking and anticipation readily realizable.

In defense of Quintus, it should be pointed out that their priority is to port their tools to a variety of platforms, rather than enhancing features. With limited resources they cannot do everything.

5.1.4 ALTERNATIVE LOGIC PROGRAM-MING IMPLEMENTATION APPROACHES. This section describes how the development could have been enhanced by a tool set that provided a higher order construct such as object orientation through ProTALK and logic extensions and object orientation through metaProlog.

5.1.4.1 ProTALK. An equivalent but more easily maintainable knowledge base definition could have been achieved in ProTALK (Quintus, 1988a), an object-oriented Prolog system. ProTALK version 0.1 as supplied with the ProWINDOWS package (Quintus, 1988b) used to build the user interface seemed to be effective when tested. However, it is unsupported, so we adopted the more conservative approach of first bringing the system up with a reliable simple representation. While the current temporary definitions are clearly inefficient in use, this did not lead to noticeable delays in normal operation of the CMLP demonstration.

ProTALK permits the dynamic instantiation of modifiable objects, a class hierarchy, and dynamic delegation. It is compatible with ProWINDOWS, which is also object oriented, and uses some of the same predicates.

In this implementation of object-oriented programming, sending an object a message is equivalent to calling a fixed procedure. Matching and execution can thus be very fast.

The objects defined in ProTALK have the same kinds of properties as the objects required for CMLP. Objects can be created and destroyed dynamically. Data in objects are stored as attribute-value associations. The values of attributes may be retrieved (this, in ProTALK, is achieved, as is all computation, by sending an object a message). Each object is a member of class, which provides it with properties. These classes are hierarchically organized. (In ProTALK,

the hierarchy provides more than data—arbitrary computational methods.) Also, pattern matching and backtracking may be used to enumerate the members of a class, as in CMLP. In addition, ProTALK provides both delegation and initialization. With the former facility, objects can delegate some of their behavior to other objects. Such delegations are dynamically modifiable. With the latter, every class has an initialization method called upon instantiation.

These capabilities would be used in five principal ways:

1. Provide an Expansible Class Hierarchy. This is a primitive of ProTALK and should therefore be efficient and reliable.

2. Simplify Data Access. The simplicity of the knowledge representation scheme used in the CMLP demonstration has certain drawbacks. One is that some inferences are "long-winded," as discussed in Section 4.2.2 under inference.

3. Speed Up Data Access. The current code for representing the values of an instance is as a list of values. To access a value for an attribute, the code steps though a list of attributes for the class in lockstep with the list of values. This process, although adequate for knowledge bases of the size and complexity of CMLP knowledge bases, could be too inefficient for a scaled-up version. The representation in ProTALK is much more efficient, since it involves representing the attribute name as a functor. Thus the search for data can use the clause-finding and argument-indexing facilities offered by Quintus Prolog. (The downside is that there are also many more terms in the database—at least one for every attribute-value pair.)

4. Simplify User Interface Code. The CMLP browsers and composite browser could use the message-passing and private data of ProTALK's objects with very desirable effects on simplifying the code and thus making it more maintainable.

5. Provide Tighter Control on Acceptability of User-Supplied Values. ProTALK provides an entry argument for each attribute, called a

"facet." Facets are used for metaknowledge and attached procedures and would be used in validation/verification processing.

5.1.4.2 MetaProlog. MetaProlog (Bowen and Weinberg, 1985) is an approach to redressing the difficulties associated with assertion and retraction. These difficulties include the fact that there is no definition of first-order proof in which the set of axioms is not fixed. This undermines the very basis of logic programming.

In metaProlog, a proof is obtained via the predicate demo:

demo(Theory,Goal,Proof)

whose correctness and completeness are specified by:

1. If demo(Theory,Goal,Proof), then Goal is derivable from Theory by Proof.
2. If Goal is derivable from Theory via Proof, then demo(Theory,Goal,Proof).

In this predicate, Theory corresponds to the set of terms and clauses defining the program in a regular Prolog program. Goal is the goal to be proved, and Proof is the proof tree, to be used for explanation or other purposes. The key difference from regular Prolog is that instead of the logic-destroying predicates assert and retract, metaProlog provides:

addto(Theory,Axiom,NewTheory)

dropfrom(Theory,Axiom,NewTheory)

Thus the proof in demo/3 is carried out on an unchanging set of axioms (i.e., theory).

MetaProlog also allows the specification of multiple theories, as in:

demo(Theory1&Theory2,Goal,Proof)

with the interpretation that if a required axiom is not found in Theory1, it will be searched for in Theory2. This allows for the hypothecation of axioms, such as:

demo([H1,H2,H3]&Theory,Goal,Proof)

where the hypothesized axioms H1, H2 and H3 will be searched before those in Theory. The programmer (or program) could amend the list of hypothesized axioms until a Proof with desired properties had been produced.

Bowen and Weinberg (1985) address other issues, including the use of qualification and relationships with parallel processes. They recognize efficiency problems, and had not, as of the writing of their paper, produced a full implementation. The work is, however, very promising in filling an important theoretical and practical gap in the rigorous application of logic programming.

MetaProlog could be very useful in design problems such as the CMLP application. It would provide a well-founded approach to the representation of system updates, and it would also manage the space of hypotheses generated and evaluated by the user. For example, in the SWaT model the user may alter various settings for priority (of target, weapon platform, and armament), number of engageable threat, and so on. The updates between a CALC within a trial would be prepended to the current (or initial) settings, thus:

demo(UpdatedValues&CurrentTheory, Goal,Proof)

As the user specified UpdatedValues by interacting with the user interface, the successive sets of changes would be retained. The user could then review and select from these changed sets when creating new ones. When the user pressed CALC, the list of contexts would be collected backwards (upwards in the hierarchy) so that the oldest changes would be last and appended to the CurrentTheory. Newer changes would then mask older changes to the same values. A new function would be defined to coalesce the updates in such a list (overriding older values as before), thus producing a saved new single update set that would then be made available to the user as before. Unfruitful branches in the space could then be pruned.

MetaProlog would have been a useful facility to have available within the Quintus Prolog environment used for implementation. However, it is not currently available as a separate module. Interpreting metaProlog concepts into Quintus Prolog as alterations to

the proof procedure code would not be efficient.

5.1.5 USER INTERFACE FOR CMLP DEMONSTRATION.

The CMLP software requirements specification developed usability requirements for the user interface. Our evaluation against these requirements is given in Table 5-1.

5.2 APPLICATION OF CONCEPTUAL MODELING AND LOGIC PROGRAMMING TO C² DESIGN

In this section we evaluate the potential of CMLP technology to support the C² process by considering how successful the conceptual representation of C² systems described in Section 3.2 would be in fulfilling the goals described in Section 3.1. For each goal we attempted to answer the following eight questions, achieving the answers summarized in Table 5-2 and discussed goal by goal below.

- *What is the priority of this goal and the CMLP features to implement it?*

- *Is use conceptually difficult?* This criterion evaluates whether the typical user will be able to understand the nature of the operations to be performed and perform them correctly.

- *Does use require difficult manipulations?* This criterion evaluates whether this particular function can be accomplished easily using a user-friendly interface.

- *Will the process be accurate and consistent?* This question evaluates whether the process can be performed in a way that will be accepted as being true by C² designers. Since many of the goals are conceptual rather than numeric, there is no "accurate" answer. Consistency deals with whether the same results will be repeatable across typical users or even by the same user from day to day.

- *What utility can be gained from this process?* This question evaluates what utility can be achieved by the C² designer if the goal is achieved, and also the expectation as to whether the goal will be achieved.

- *Will CMLP use provide more convenient analysis?* This question evaluates whether

the C² designer will be able to more easily achieve the desired outputs.

- *Will CMLP use provide a better analysis product?* This question evaluates whether an analyst can do a better job with CMLP, independent of the time that it takes him to do the job.

- *Is experimentation needed to improve this evaluation?* Many of the answers to the first six questions will be uncertain. This question will therefore evaluate whether the CMLP design team would recommend a program of further experimentation to improve the design product.

1. Analyze Performance by Computation.
Goal 1 pertains specifically to the potential of the SWaT model described in Section 3.2.2. Concepts for the SWaT model range all the way from simple rules of thumb to detailed evaluations that consider location of sensors, weapons and threats. A major concern is estimating the effect of C² systems. While we have been able to incorporate some C² rules of engagement in the demonstration system SWaT model, it is extremely difficult to provide the degradation caused by less than perfect C². The demonstration SWaT model does not include a temporal analysis and hence cannot predict degradation due to response time or evaluate concepts such as shoot-look-shoot effectively. Also, the demonstration system model includes only the capability to deal with the first attack. It may be possible to include consideration of waved attacks, and if this is done the model should also estimate defensive system losses to allow evaluation of resources available for the second wave.

SWaT model performance may be improved by using simulations describing a variety of threat attacks to anchor a mathematical model. Even without a simulation, the user may correct an output from a CMLP model based upon any data that he may have on expected system performance.

Table 5-1. Evaluation of User Interface

1. Completeness of Coverage

Criterion: The designer should have at his disposal all information necessary to assess and apply the repertoire of established system enhancements, or to enter new ones.

Evaluation: Excellent

Comment: The baseline database is large and consists of many disparate types of data. It is broken down by element and high-level summaries are provided. Data items are formatted in user terms, rather than reflecting the internal organization of the knowledge bases. Provision is made, via browsers, for the display of arbitrarily large numbers of system components. The displays of system components are grouped in a manner both meaningful to the user and within the context of the sequential usage of CMLP (e.g., the definition of range bins on the sensor form).

2. Symbolic Interaction

Criterion: To maximize the utility of the design aid, the user should be required to perform a minimum of direct programming. That is, the objects and relations that constitute the system should be accessible in symbolic form at the appropriate level of abstraction.

Evaluation: Excellent

Comment: The user is never required to know anything of the internal representation in CMLP and is never required to program. The knowledge representation scheme is completely transparent to the user, who thinks merely in terms natural to one designing C^2 systems.

3. Explanation

Criterion: It is desirable for the system to be able to explain its actions and recommendations. This will be restricted in the CMLP demonstration to permitting the user to review selected rules that result in an inference.

Evaluation: Good

Comment: Explanation facilities are limited in the manner stated.

4. Consistency of Interface

Criterion: The means of interacting with the symbolic components of the interface should be consistent across type and across function.

Evaluation: Excellent

Comment: The behavior of buttons, menus, and all other means of interacting with symbolic components is completely uniform across the entire user interface. The only exceptions occur when the pragmatics of providing a simple and natural user interface outweighed the dictates of formal requirements. For example, in the SWaT model selecting a line in the modifiable browsers leads to a MODIFY. In the main, other browsers allowing modification also allow other actions, and so require election plus the use of a specific button. Where a single MODIFY application-specific application button is used, this is to maintain consistency within the form.

5. Mixed Initiative Dialog

Criterion: The system should do what it does best; the designer should concentrate on his design problems. The system should therefore assume the burden for all activities which it can

Table 5-1. Evaluation of User Interface (continued)

support without input from the user; this amounts to all of the housekeeping functions. The system should interact with the user in a way that exchanges responsibility for directing the man-machine dialog smoothly and in terms of which entity — the user or the computer — can most efficiently carry out a processing step.

Evaluation: Excellent

Comment: The user is never required to do something the system could. Because of the nature of the C^2 system design process, CMLP does not do a great deal of stand-alone inferencing (unlike the stereotypical expert system), but where inferences can be made (as in impact assessment) they are. Most decisions and judgments are made by the user. Almost all the information that could be desired by the user is usually zero, one, or two mouse-clicks away.

6. **Progressive Disclosure**

Criterion: The designer should not be overcome with too much data when what he wants is information. Information should be portrayed at a variable level of granularity, and the designer should be able to vary this level with his objectives. Information should be prioritized and displayed in order.

Evaluation: Excellent

Comment: Progressive disclosure is used extensively throughout the user interface, starting with the summary. The user needs only to click in the relevant box and the detailed form becomes available. Within a form, the progressive disclosure is used widely. For example, when the user clicks on a weapon platform in the weapon specification form, its armaments appear; and when the user clocks on an armament, its probability of kill against the targets appears.

7. **Communication**

Criterion: Members of the C^2 system design team should be able to share information. thoughts, and hypotheses using the system as a medium.

Evaluation: Good

Comment: CMLP has been designed for a single user at a time, but provision is made for storing comment fields with the knowledge bases. The tool can make knowledge bases accessible to other users and thus facilitate communication. No inter-user mail capability or user group system will exist, but the SunView mail-tool may be run independently within the CMLP windowing environment.

8. **Training**

Criterion: A basic "help" subsystem is provided.

Evaluation: Good

Comment: Context-sensitive help is available inside most of the boxes of each form. In future versions, help tailored to the skill level of the user could be provided. Since ProWINDOWS views do not generate an entry_message, passage of the mouse into them is not detectable, so the help system does not operate for windows such as description or comment windows.

9. **Adaptivity**

Criterion: The system should reflect the goals, skills, and background of the user. It should also change as the user develops greater facility for interacting with the system. A certain amount of personal customization of the interface is also useful.

Table 5-1. Evaluation of User Interface (continued)

Evaluation: This has not been provided in the CMLP demonstration system.

Comment: There are no provisions within the delivered CMLP for user adaptivity, although "hooks" have been provided and the knowledge representation system is adequate to represent different user types.

Table 5-2. Evaluation of CMLP Goals

1. **Analyze Performance by Computation**

   Priority?   High

   Conceptually Difficult?   Limited model may force innovative use

   Require Difficult Manipulations?   No

   "Accurate" and Consistent?
   - Difficult to include C^2 effect except by simulation
   - Questionable whether CMLP models can be anchored
   - Possible that if CMLP requires user to "correct" conclusions, he will lack confidence in results

   What Utility?

       Potential Benefits: Can provide rule-of-thumb to reduce need for simulation-based evaluation

       Achievability: In some form

   More Convenient?   Yes

   Provide Better Product?   Yes, if user understands process

   Further Experimentation Needed?   Try different levels of detail in SWaT model, perform experiments on other systems

2. **Analyze Performance by Simulation**

   Priority?   Very high, but may be too expensive

   Conceptually Difficult?   No

   Require Difficult Manipulations?   Defining complete set of simulations and interpreting data are difficult

   "Accurate" and Consistent?   Should be more accurate than current manual approaches

   What Utility?

       Potential Benefits: Provides accurate techniques for evaluating C^2 performance

       Achievability: To be determined

   More Convenient?   High potential

   Provide Better Product?   Very high potential

   Further Experimentation Needed?   Yes; may be very high cost

Table 5-2. Evaluation of CMLP Goals (continued)

3. **Analyze C^2 Performance by Goal Scoring**

   Priority?   High

   Conceptually Difficult?   No

   Require Difficult Manipulations?   No

   "Accurate" and Consistent?   A major concern; see text

   What Utility?

   > *Potential Benefits:*   Provides alternative to simulation, forces thoughts on concepts for improve C^2, uses natural cognitive techniques
   >
   > *Achievability:*   Excellent

   More Convenient?   Yes

   Provide Better Product?   Yes

   Further Experimentation Needed?   Experiment with multiple users, one user over time, levels of self-defined goals

4. **Analyze C^2 Capacity Constraints**

   Priority?   Medium

   Conceptually Difficult?   No

   Require Difficult Manipulations?   No

   "Accurate" and Consistent?   Yes

   What Utility?

   > *Potential Benefits:*   Helps user define and understand capacity issues
   >
   > *Achievability:*   May be difficult to define models (e.g., computers with spaghetti code)

   More Convenient?   Yes

   Provide Better Product?   Yes

   Further Experimentation Needed? Try to develop capacity models for processor computations, memory, C^2 operators, communications links, I/O devices

5. **Analyze System Life-Cycle Costs**

   Priority?   High

   Conceptually Difficult?   No

   Require Difficult Manipulations?   No

   "Accurate" and Consistent?   Yes

   What Utility?

   > *Potential Benefits:*   Combine costs with other evaluation data
   >
   > *Achievability:*   Excellent

Table 5-2. Evaluation of CMLP Goals (continued)

More Convenient? Yes; cost data parameters can be estimated from other C^2 systems

Provide Better Product? Yes

Further Experimentation Needed? No

6. **Identify Effect of Changes in Other Parts of System**

Priority? High

Conceptually Difficult? No

Require Difficult Manipulations? No

"Accurate" and Consistent? Yes, but new rules may be needed for some systems, may want to limit number of rules

What Utility?

   Potential Benefits: Provides more complete identification of change
   Achievability: Excellent

More Convenient? Very large rule set may be needed to cover a wide variety of systems, and user will have to understand and evaluate each rule

Provide Better Product? Yes

Further Experimentation Needed?
   - Many C^2 systems
   - Level of detail/threshold for including rules

7. **Define Options Based on Similar Situations**

Priority? High

Conceptually Difficult? No

Require Difficult Manipulations? No

"Accurate" and Consistent? Yes

What Utility?

   Potential Benefits: Provides user with concepts "proven" on other systems
   Achievability: Excellent

More Convenient? Yes

Provide Better Product? Yes

Further Experimentation Needed? Proof-of-concept experiments

8. **Evaluate Options Based on Similar Situations**

Priority? Medium

Conceptually Difficult? No

Require Difficult Manipulations? No

Table 5-2. Evaluation of CMLP Goals (continued)

"Accurate" and Consistent?   Yes

What Utility?

 Potential Benefits: Provides "experience" data from other systems to perform evaluation/provide data

 Achievability: Excellent

More Convenient?   Yes

Provide Better Product?   Yes

Further Experimentation Needed?   Proof-of-concept experiments

9. **Identify Requirements/Design Dependence**

Priority?   Medium

Conceptually Difficult?   No

Require Difficult Manipulations?   No

"Accurate" and Consistent?   Depends on knowledge base

What Utility?

 Potential Benefits: Identifies which design features are not needed

 Achievability: Many cases too complex

More Convenient?   Requires very large database

Provide Better Product?   Yes

Further Experimentation Needed?   Develop, populate, and use a traceability model

10. **Collect Data**

Priority?   High

Conceptually Difficult?   No

Require Difficult Manipulations?   No

"Accurate" and Consistent?   Yes

What Utility?

 Potential Benefits: Organizes and makes though processes consistent, should enhance creativity

 Achievability: Excellent

More Convenient?   Yes, but may require user to collect more data than he is used to, and than he needs

Provide Better Product?   Yes

Further Experimentation Needed?   Measure user's reaction to process

Table 5-2. Evaluation of CMLP Goals (continued)

11. Identify Needed Data

Priority?   High

Conceptually Difficult?   No

Require Difficult Manipulations?   No

"Accurate" and Consistent?   Should be

What Utility?

> *Potential Benefits:*   Ensures that user considers relevant data
> *Achievability:*   Yes, through two alternative approaches:
> - CMLP design incorporates multiple levels of detail
> - CMLP accepts available data, evaluates if each analysis can be performed

More Convenient?   Yes

Provide Better Product?   Yes

Further Experimentation Needed?   Evaluate alternative approaches

12. Provide Data From Knowledge Bases

Priority?   High

Conceptually Difficult?   No

Require Difficult Manipulations?   No

"Accurate" and Consistent?   Yes

What Utility?

> *Potential Benefits:*   Saves user from hunting down data by providing "similar" data
> *Achievability:*   Affected by hardware issues

More Convenient?   Yes

Provide Better Product?   Yes

Further Experimentation Needed?   Evaluate whether CMLP can identify and control similar data

13. Check Data Against Own Knowledge Base

Priority?   Low

Conceptually Difficult?   No

Require Difficult Manipulations?   No

"Accurate" and Consistent?   Yes

What Utility?

> *Potential Benefits:*   Improves data quality
> *Achievability:*   Excellent

Table 5-2. Evaluation of CMLP Goals (continued)

More Convenient?   Most users prefer using tools to search and present data, rather than reviewing data entered

Provide Better Product?   Yes

Further Experimentation Needed?   No

14. **Evaluate Change to System**

Priority?   Medium

Conceptually Difficult?   No

Require Difficult Manipulations?   No

"Accurate" and Consistent?   Should be

What Utility?

    Potential Benefits: Aids in evaluating current performance and design concepts by organizing approach, providing evaluation data

    Achievability: C^2 performance evaluation is satisfactory only with simulation interface; multiple evaluations are possible

More Convenient?   Yes

Provide Better Product?   Yes

Further Experimentation Needed?   Variety of evaluation techniques, different user

15. **Compare Changes to System**

Priority?   High

Conceptually Difficult?   No

Require Difficult Manipulations?   No

"Accurate" and Consistent?   Should be

What Utility?

    Potential Benefits: Allows evaluation of alternative fixes

    Achievability: C^2 evaluation is the key, clearly achievable with simulation

More Convenient?   Yes

Provide Better Product?   Yes

Further Experimentation Needed?   Variety of evaluation techniques, different users

16. **Evaluate C^2 Performance**

Priority?   High

Conceptually Difficult?   No

Require Difficult Manipulations?   No

"Accurate" and Consistent?   Unknown

Table 5-2. Evaluation of CMLP Goals (continued)

What Utility?

    Potential Benefits: Allows designer to measure C^2 design concepts, status C^2 performance

    Achievability: Through three alternative concepts: simulation, improved SWaT models, goal evaluation

More Convenient? Yes

Provide Better Product? Depends on concept selected

Further Experimentation Needed? Experiment with goal assessment, SWaT improvements, development of simulation and interface

17. **Evaluate Deployment Strategies**

    Priority? Medium/low

    Conceptually Difficult? No

    Require Difficult Manipulations? No

    "Accurate" and Consistent? Yes

    What Utility?

        Potential Benefits: Allows comparison of alternative deployment strategies based on funding constraints

        Achievability: May be difficult owing to C^2 evaluation issues

    More Convenient? Yes

    Provide Better Product? Yes

    Further Experimentation Needed? Of low priority·

18. **Compare Design Changes Across Systems**

    Priority? Medium/low

    Conceptually Difficult? No

    Require Difficult Manipulations? No

    "Accurate" and Consistent? Unknown

    What Utility?

        Potential Benefits: Allows prioritization of system developments

        Achievability: May require different model; see text

    More Convenient? Unknown

    Provide Better Product? Unknown

    Further Experimentation Needed? Development of concept

Use of a SWaT model should save very much time, even if a detailed system simulation is available, in evaluating defensive system response to projected changes in threat. The SWaT model can suggest not only changes needed in existing sensor and weapon systems, but also new concepts of engagement, such as engaging the threat earlier in order to bring more assets to bear.

Evaluating the effectiveness of a SWaT modeling concept would require experimenting with incorporating different levels of detail in the model and with some source of system truth, such as a simulation. We recommend that this experiment and analysis be performed.

2. Analyze Performance by Simulation. A SWaT model that includes consideration of time is equivalent to a simulation. Goal 2 deals with creating a simulation interface within the conceptual modeling system so that simulation can be run and its output data interpreted. The simulation interface would have to generate all of the inputs associated with simulations. When evaluating concepts for a design change, this may entail generating the simulation model of the design change to the system. Also, we note that it is difficult to determine in advance what parameters may affect a particular evaluation problem and to generate a simulation plan that will vary all of them. Another area of concern is interpretation of simulation results, particularly when a large number of parameters are involved.

A CMLP tool capable of driving a simulation can perform fast conceptual model evaluations when that is appropriate and periodically validate those evaluations by detailed simulations. This process should aid the user in understanding the C^2 design issues and should lead to highly accurate results. This process is therefore expected to have a very high payoff as a CMLP concept.

Further evaluation of the simulation interface concept is needed to establish proof of principle. The experiment requires not only a CMLP tool, but also a flexible simulation

that includes C^2 detail. To effectively incorporate a knowledge base-driven simulation plan and setup, the simulation should be organized in a pure object-driven sense. Also, the simulation should include clear measures of performance that can be analyzed by the CMLP tool to produce a measurement of interest to the CMLP user. This experimentation is likely to be of high cost.

3. Analyze C^2 Performance by Goal Scoring. Since concepts suggested by goal 1 are not likely to evaluate C^2 very well, and concepts associated with goal 2 are likely to be expensive and difficult to use, an alternative form of C^2 system evaluation is of high priority. Goal 3 addresses the potential of developing a goal evaluation model that can use both generic and specific goals for a particular C^2 system. The concept is attractive in that goal evaluation is a common cognitive evaluation process. A SWaT model can provide an indication of perfect C^2 performance, and a goal evaluation system may indicate the nature of degradation associated with the C^2 system. A side product of goal evaluation is that identifications of unsatisfied goals should directly suggest improvements to the C^2 system.

The major issue associated with goal evaluation is that it is subjective and hence likely to produce different results when used by different people. This may tend to make the results of this appro unacceptable to the C^2 community. On the other hand, the goal evaluation model may be used to expose such differences, make them explicit, and point the way to research and resolve them. Goal evaluation can also generate new ideas for C^2 designers through comparison to other systems. For example, air defense weapon assignment and weapon control are largely manual operations performed in sectors, with overlap in responsibilities and providing highly nonoptimal results. Computer, man-machine interface, and algorithmic capability being developed for the Strategic Defense Initiative could greatly improve the use of interceptor and missile systems against air

defense attack. Use of a CMLP goal evaluation model would help focus on these issues. Investigation into the use of goal-based evaluation is thus highly desirable. The investigation should include experiments employing multiple users and experiments employing one user over time in slightly different circumstances, and should permit the user to define his own goals.

4. Analyze C^2 Capacity Constraints. Physical capacity constraints include computer processing speed, computer memory size, the amount of work that a person or group of people can do, and the amount of data that can be passed (1) across a communications link, (2) from a manned input device (terminal) into a computer system through communications equipment, or (3) through computer interfaces. The load on a capacity-limiting device can usually be predicted by simple mathematical formulas operating upon the number of instances to be modeled over a fixed period of time. The instances include elements of threat, the number and types of reporting sensors, and the numbers and types of weapons to be controlled. While the loading is likely to behave according to simple mathematical processes, the processes may be very difficult to define from available but very complex inputs, such as the computer software listing.

Experimentation is highly desirable through generating a variety of capacity models from alternative input sources. Additional experimentation is needed to determine whether there are ranges of input for which the loading is not easily predictable.

5. Analyze System Life-Cycle Costs. Cost and units of fiscal-year cost are the major constraints associated with development of C^2 systems. Therefore it is extremely important that the conceptual model have the ability to compare cost against other performance measures. A conceptual modeling process that incorporates cost along with other factors may provide a uniform database for evaluation. As CMLP cost models are developed for multiple C^2 systems, data may

be interchanged, providing better and more convenient data generation.

6. Identify Effect of Changes in Other Parts of System. The CMLP system should have the capability of cuing the user to potential model changes associated with one or more changes being evaluated in other areas. A concept for implementing this change is discussed in Section 3.2.8, Systems Sensitivity Model. The benefits gained from a system sensitivity model will depend on the rules established within the model. The model will need a large database of routine rules that are fairly obvious. However, to be effective it will also need to suggest to the user things that he may not have considered. For example, one air defense concept for dealing with more concentrated attacks (or higher threat density) is to extend the geographic range under which the threat can be engaged, and then send out the response early in the attack. This type of change will not necessarily be obvious to someone who has been working on the system for a long time and who will thus tend to use establish procedures within the system.

The system sensitivity concept should have a high payoff. The major issue is the number of rules that should be included within a generic model. The threshold on whether to include a particular rule is based upon the probability that the rule is likely to represent a C^2 system sensitivity. As the threshold is lowered, the CMLP system will suggest many more areas of change that the user must evaluate. As the threshold is raised, fewer rules will be included within the CMLP system and significant areas may fail to be identified for a particular analysis. Thus evaluation of the level of detail of sensitivity rules is a high-priority experiment for furtherance of CMLP technology.

7. Define Options Based on Similar Situations. Two goals are associated with the concept of analogy development, discussed in Section 3.2.9, which allows users to benefit from the experience of previous system users. In the first, the CMLP system can use analogy

development to find possible fixes for a particular problem by searching its knowledge bases for other C^2 systems that have had similar problems. Assume, for example, that a user expected enemy forces to be able to overrun a defensive system he was evaluating, i.e., that we could not bring enough weapons to bear to protect a high percentage of our assets. If proposed SDI system concepts were modeled in a CMLP database, the air defense user might be able to extract from that database the SDI concept of adaptive preferential defense. That concept analyzes the assets within one class to determine which are most likely to be attacked and then concentrates the defense on those assets. Clearly this can be done much more accurately for strategic defense, where the threat follows a ballistic trajectory, than in an air defense system, where the bomber and cruise missile threats follow a devious trajectory to their targets. Nevertheless, the concept may offer some benefit in the air defense world. The payoff for analogy development of this type would be maximized if the CMLP system stored all uses of the tool, regardless of the user's assessment of the results.

8. Evaluate Options Based on Similar Situations. The second goal concerned with analogy development relates to evaluating the magnitude of a design change's effect by examining the effects of similar changes in other C^2 systems. Again, the utility of analogy development is greatly enhanced if a large amount of data is stored by the previous users. An issue associated with analogy development is that the data previously stored may not be validated data in the sense that they may not be considered accurate by the current user (or, for that matter, by the original user). Also, it is highly likely that the new user will need to verify the validity of the analogy made by the CMLP tool. Nevertheless, analogy development is an important CMLP concept requiring proof-of-concept experiments.

9. Identify Requirements/Design Dependence: Conceptual modeling is an appropri-

ate way to develop requirements-to-design traceability models that allow the user to infer a possible design change as requirements change. We see two impediments to achieving this goal. The first is that a mapping of requirements to design functions for complex C^2 systems may involve an extremely large database. The second is that the design may be very interconnected.

The previously cited F-111 example serves to illustrate this point. The requirement to land on sand beaches led to engine inlet flaps to prevent the ingestion of sand and to very large tires to support the plane on sand. While the inlet flaps would be directly traceable to the sand-landing requirement using a requirements-to-design tool, the situation for the tires would not be traced so easily. Tire size is directly related to a large number of factors, including the pressure needed to bear the aircraft weight, the tread design needed to stabilize tire contact with the runway, acceleration and stopping distances, and, finally, the requirement to land on sand. Thus it may be difficult to arrive at the suggestion to change to smaller tires based on using a requirements-to-design traceability model, but easy based on comparing the F-111 design to that of another aircraft.

Experiments on the utility of the requirements-to-design traceability model are needed if this capability is to be included in the conceptual model. We believe this to be of lower priority than the experiments suggested in other areas.

10. Collect Data. One major benefit of the conceptual modeling process will be the convenient handling of data to support command and control designer analysis. The process of organizing data through a carefully designed knowledge base will greatly benefit the designer in organizing his own thoughts on the C^2 system. However, this may force the user to locate and input more data than he feels is warranted by the process that he is working on. One reason is that he may never before have used all of the relevant data. Another is that the fact that it is generic may

cause the CMLP model to ask for more data than needed. We believe that the data organization process will greatly benefit C^2 designers but see the need for further experimentation to validate that principle for a variety of alternative users.

11. Identify Needed Data. This goal relates to having the CMLP model help the user identify data he needs. For a SWaT model expanded to allow more accuracy by consideration of alternative effects (e.g., geographical area, temporal, environmental), the user may need to provide additional data such as how many of our interceptors we might lose in the process of attacking each type of enemy air threat, and also provide all of the parameters associated with a second-wave attack. In the cost area, a particular analyst may want to include detailed life-cycle cost terms such as the cost of maintaining an interceptor aircraft in inventory, and may be given the option of evaluating this cost based on historical data in the database or personalizing it to develop the distinction between cost associated with a single-pilot F-16 and a dual-crew F-15.

There are two alternative concepts for implementing this goal. One is that the CMLP design be developed with multiple levels of detail to support different analysis processes. This means that as the user populates the database, he must tell the CMLP model what he wants to do with the database, and then the model can tell him what data he needs. The alternative approach is to have the CMLP system attempt the data analysis after the user has input the data readily available to him. In this approach the CMLP tool must identify which analyses cannot be reliably performed and alert this fact to the user so that he can either provide more data or decide not to rely on a particular analysis. Further research is needed to evaluate the two concepts.

12. Provide Data From Knowledge Base. Considerable user time can be saved and consistency improvements achieved if the CMLP tool incorporates a capability of providing data from the knowledge bases established in previous analyses. This capability might be provided by having the system search its knowledge base and provide data to the user for review before storing the information in a particular baseline system. Experimentation is needed to evaluate the approach and determine the utility of data from analysis of other C^2 systems.

13. Check Data Against Own Knowledge Base. Providing the CMLP tool with a capability to check data against data already in its knowledge base has similar payoffs to those of goal 12. The approach would be to have the CMLP system check user-supplied data against other data stored during previous sessions. Experimentation similar to that recommended for goal 12 is recommended.

14. Evaluate Change to System. Goal 14 is the first of five relating to the end purpose of the CMLP user in a particular session or group of sessions. Goal 14 specifically relates to an an absolute evaluation of the impact of a system change. We judged this goal to be of only medium priority because it is not clear that an absolute measure is needed. Achieving this goal will depend on how well the system can perform the simulation and other evaluations discussed for the first five goals. The difficulty is that the tool will produce an evaluation that incorporates many factors; the evaluation will itself require evaluation. This issue requires research using a variety of evaluation techniques. Involving different users in the experimentation process will determine the sensitivity of results to particular users.

15. Compare Changes to System. Goal 15 relates to comparing alternative changes within a system. This capability is the key to CMLP success because it would allow the user to evaluate alternative approaches (fixes) to an undesirable situation and justify changes to C^2 rather than requiring more weapons or better sensors. For example, a C^2 designer may wish to use the CMLP tool to identify C^2 improvements to deal with a projected threat

increase, as an alternative to very expensive sensor and weapon improvements. Such a C^2 improvement might be achieved by more effective weapon-target assignment algorithms or by the capability to tell and use data from all available sensors to reduce dependence on sensor survivability. Displaying the amount of comparative data that might be desired may be a problem. The demonstration system showed many benefits of a windowing interface but was not designed to allow the user to look at the results of two or more evaluations at the same time. In fact, we found it difficult to provide enough room for even identifying all of the data needed in conceptual modeling. A large number of experiments are required on a variety of C^2 systems using different evaluation techniques such as SWaT models, simulations, and goal scoring.

16. Evaluate C^2 Performance. Goal 16 relates to the capability to evaluate C^2 system performance. This capability is of high priority as it allows absolute evaluation of C^2 design concepts. Absolute measures of performance may be needed to allow comparison of changes from system to system, such as might be required in formulating a position on which of two or more C^2 systems more urgently needs funding in the near term (assuming goal 18 is not satisfied). Issues include defining the measures of performance and selecting simulation, SWaT, or goal-scoring approaches for specific design tasks. Extensive experimentation is required in this area.

17. Evaluate Deployment Strategies. Goal 17 relates to the evaluation of deployment strategies in order to make the best defensive system available at each deployment stage subject to fiscal constraints. This is of medium or low priority since there may be easier ways to approach the problem and deployment plans are highly constrained, reducing choices. Data collected in association with the evaluation of goal 14 will help determine the capability of the CMLP system to support the present goal.

18. Compare Improvements Across Systems. Goal 18 deals with the need for comparison of design changes across a variety of systems. This capability could be used by funding control agencies to set development priorities. We do not believe that this is the most likely use of the CMLP tool. Comparative evaluation is difficult in that measures of performance on one system rarely relate directly to measures of performance on another. In addition, comparative evaluation does not facilitate consideration of the overall defense posture. A different conceptual model might be needed to approach this goal. Such a model would have to recognize the overall force structure and defense posture of the particular organization, and incorporate this posture in the evaluation process. A concept for doing so is the mission-oriented analysis that used by NATO for evaluating command and control systems (see Signori and Starr, 1987).

BIBLIOGRAPHY

Apple Computer, Inc. (C. Rose, et al.), *Inside Macintosh*. Reading, Mass.: Addison-Wesley, 1985.

Barr, A., and E. A. Feigenbaum, *The Handbook of Artificial Intelligence*. Los Altos, California: Kauffmann, 1981.

Bateson, G., *Mind and Nature*. New York: Bantam, 1979.

Battani and Meloni, *Interpreteur du Language de Programmation PROLOG*, Research Report, Artificial Intelligence Group. Luminy, France: University of Aix-Marseille, 1973.

Bendl, J., P. Koves, and P. Szeredi, *Proceedings of Logic Programming Workshop*. Debrecan, Hungary, 1980.

Bobrow, D., L. DeMichiel, R. Gabriel, S. Keene, G. Kiczales, and D. Moon, *Common Lisp Object System Specification*. X3J13 Document 88-002R, 1988.

Bordiga, A., J. Mylopoulos, and H. Wong, "Generalization/Specialization as a Basis for Software Specification." In M. L. Brodie, J. Mylopoulos, and J. W. Schmidt (eds.), *On Conceptual Modeling*. New York: Springer-Verlag, 1984.

Bowen, K., and T. Weinberg, *A Meta-Level Extension of Prolog*, Technical Report CIS-5-1. Syracuse, N.Y.: Syracuse University School of Computer and Information Science, May 1985.

Brodie, M. L., "On the Development of Data Models." In M. L. Brodie, J. Mylopoulos, and J. W. Schmidt (eds.), *On Conceptual Modeling*. New York: Springer-Verlag, 1984.

Brodie, M. L., J. Mylopoulos, and J. W. Schmidt (eds.), *On Conceptual Modeling*. New York: Springer-Verlag, 1984.

Bruynooghe, M., "Memory Management of Prolog Implementations," *Proceedings of Logic Programming Workshop*. Debrecen, Hungary, 1980.

Bylander, T., and B. Chandrasekaran, "Generic Tasks in Knowledge-Based Reasoning." In B. R. Gaines and J. H. Boose (eds.), *Knowledge Acquisition for Knowledge-Based Systems*. New York: Harcourt Brace Jovanovich, 1988.

Church, A., "The Calculi of Lambda-Conversion," *Annals of Mathematical Studies No. 6*. Princeton, N.J.: Princeton University Press, 1941.

Clark, K., and F. McCabe, "IC-Prolog—Language Features," *Proceedings of Logic Programming Workshop*. Debrecan, Hungary, 1980.

Clocksin, W. F., and C. S. Mellish, *Programming in Prolog*. New York: Springer-Verlag, 1984.

Colmerauer, A., H. Kanoui, R. Passero, and P. Roussel, *Une Systeme de Communication Homme–Machine en Francaise*, Research Report, Artificial Intelligence Group. Luminy, France: University of Aix-Marseille, 1973.

Common Windows Manual. Mountain View, Calif.: IntelliCorp, Inc., 1986.

Darvas, P., "Logic Programming in Chemical Information Handling," *Proceedings of Logic Programming Workshop*. Debrecen, Hungary, 1980.

Feigenbaum, E., "The Art of Artificial Intelligence: Themes and Case Studies of Knowledge Engineering," *Proceedings of the Fifth International Joint Conference on Artificial Intelligence, 1977*, pp. 1014–1029. Pittsburgh: Department of Computer Science, Carnegie-Mellon University, 1977.

Fikes, R. E., and G. Hendrix, "A Network-Based Knowledge Representation and Its Natural Deduction System," *Proceedings of the Fifth International Joint Conference on Artificial Intelligence, 1977,* pp. 235–246. Pittsburgh: Department of Computer Science, Carnegie-Mellon University, 1977.

Gaines, B. R., "An Overview of Knowledge Acquisition and Transfer." In B. R. Gaines and J. H. Boose (eds.), *Knowledge Acquisition for Knowledge-Based Systems.* New York: Harcourt Brace Jovanovich, 1988.

Gaines, B. R., and J. H. Boose (eds.), *Knowledge Acquisition for Knowledge-Based Systems.* New York: Harcourt Brace Jovanovich, 1988.

Gandee, P. L., M. D. Gray, and R. Sweet, "Evaluating Alternative Air Defense Architectures," *Signal,* January 1987.

Goldberg, A., and D. Robson, *Smalltalk-80: The Language and Its Implementation.* Reading, Mass.: Addison-Wesley, 1983.

Goodman, D., *Hypercard Developers' Handbook.* New York: Bantam, 1988.

Goodman, N., *Fact, Fiction, and Forecast.* Indianapolis: Bobbs-Merrill, 1973.

Green, C. C., *The Application of Theorem Proving to Question-Answering Systems,* Report No. CS 138 A1 Memo-96. Stanford, Calif.: Stanford University Press, 1969.

Hayes, P., "Computation and Deduction," *Proceedings of the Second Symposium on Mathematical Foundations of Computer Science,* pp. 105–118. Czechoslovak Academy of Sciences, 1973.

Hayes-Roth, F., D. A. Waterman, and D. Lenat (eds.), *Building Expert Systems.,* Reading, Mass.: Addison-Wesley, 1983.

Hewitt, C., and P. deJong, "Open Systems." In M. L. Brodie, J. Mylopoulos, and J. W. Schmidt (eds.), *On Conceptual Modeling.* New York: Springer-Verlag, 1984.

Hogger, C. J., *Introduction to Logic Programming.* London: Academic Press, 1984.

Jensen, K., and N. Wirth, *Pascal User Manual and Report.* New York: Springer-Verlag, 1978.

Johnson, P. E., I. Zaulkeman, and S. Garber, "Specification of Expertise." In B. R. Gaines and J. H. Boose (eds.), *Knowledge Acquisition for Knowledge-Based Systems.* New York: Harcourt Brace Jovanovich, 1988.

KEE User's Guide. Mountain View, Calif.: IntelliCorp, Inc., 1986.

Keene, S., *Object-Oriented Programming in COMMON LISP: A Programmer's Guide to CLOS.* Reading, Mass.: Addison-Wesley, 1988.

Kernighan, B., and D. Ritchie, *The C Programming Language.* Englewood Cliffs, N.J.: Prentice-Hall, 1978.

Kornell, J., "Formal Thought and Narrative Thought in Knowledge Acquisition." In B. R. Gaines and J. H. Boose (eds.), *Knowledge Acquisition for Knowledge-Based Systems.* New York: Harcourt Brace Jovanovich, 1988.

Kowalski, R., "Predicate Logic as a Programming Language," *IFIP Congress,* pp. 569–574. Stockholm: International Federation of Information Processing, 1974.

———, "Logic for Problem Solving." New York: North-Holland, 1979.

———, "Algorithm = Logic + Control," *Communication of the ACM,* Vol. 22, pp. 424–431, 1979.

Littman, D. C., "Modeling Human Expertise in Knowledge Engineering: Some Preliminary Observations." In B. R. Gaines and J. H. Boose (eds.), *Knowledge Acquisition for Knowledge-Based Systems.* New York: Harcourt Brace Jovanovich, 1988.

Logicon, Inc. (G. Silva, D. Dwiggins, and C. Montgomery), *A Knowledge-Based Automated Message Understanding Methodology for an Advanced Indications System*. RADC-TR-79-133, Rome Air Development Center AD# A072 395, 1977.

———, *Operational Concept Document for Conceptual Modeling Via Logic Programming*, OSD: W-R88-02. Woodland Hills, Calif.: Logicon, Inc., Operating Systems Division, January 1989.

———, *Interim Technical Report for Conceptual Modeling Via Logic Programming*. Woodland Hills, Calif.: Logicon, Inc., Operating Systems Division, March 1989a.

———, *Software Requirements Specification for Conceptual Modeling Via Logic Programming*, OSD: W-R88-03. Woodland Hills, Calif.: Logicon, Inc., Operating Systems Division, March 1989b.

———, *Software Top Level Design Document for Conceptual Modeling Via Logic Programming*. Marina del Rey, Calif.: Logicon, Inc., Operating Systems Division, March 1989c.

Markusz, "Application of Prolog in Designing Many-Storied Dwelling Houses," *Proceedings of Logic Programming Workshop*, pp. 249–260. Debrecen, Hungary, 1980.

Martin, F. F., *Computer Modeling and Simulation*. New York: John Wiley and Sons, 1968.

McCarthy, J., "History of LISP," *ACM SIGPLAN Notices*, Vol. 18, No. 8, 1978.

McCarthy, J., and P. J. Hayes, "Some Philosophical Problems from the Standpoint of Artificial Intelligence." In D. Michie and B. Meltzer (eds.), *Machine Intelligence 4*. Edinburgh: Edinburgh University Press, 1969.

McDermott, D., "The Prolog Phenomenon," *SIGART Newsletter*, Vol. 73, pp. 19–20, 1980.

Mellish, C., "An Alternative to Structure-Sharing in the Implementation of a Prolog Interpreter," *Proceedings of Logic Programming Workshop*, pp. 21–32. Debrecen, Hungary, 1980.

Minsky, M., "A Framework for Representing Knowledge." In P. Winston (ed.), *The Psychology of Computer Vision*. New York: McGraw-Hill, 1975.

MProlog Reference Manual. Logicware Inc., 1986.

Newell, A., and H. Simon, "The Logic Theory Machine," *IRE Transactions on Information Theory*, Vol. 2, pp. 61–79, 1956.

———, *Human Problem Solving*. Reading, Mass.: Addison-Wesley, 1972.

Noah, W. W., and S. Halpin, "Adaptive User Interfaces for Planning and Decision Aids in C³I Systems," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 16, No. 6, pp. 909–918, November/December 1986.

Noah, W. W., and R. Hopf-Weichei, *Shared Conceptual Model Development: Implications for Communications of a Cognitive Analysis of Operations*, Paper Prepared for U.S. Army Research Institute for the Behavioral and Social Sciences, OSD: W-N85-06, 1985.

O'Keefe, R. A., Articles 1682, 1704 on Prolog Bulletin Board, 1989.

Piaget, J., *Biology and Knowledge*. Englewood Cliffs, N.J.: Prentice-Hall, 1972.

Prolog Reference Manual. BIM, 1988.

Quillian, M. R., "Semantic Memory." In M. Minsky (ed.), *Semantic Information Processing*. Cambridge, Mass.: MIT Press, 1968.

Quintus, *Quintus ProWINDOWS Manual*. Mountain View, Calif.: Quintus Computer Systems, Inc., 1988a.

———, *ProTALK Programmer's Guide*. Mountain View, Calif.: Quintus Computer Systems, Inc., 1988b.

———, *Quintus Prolog Reference Manual*. Mountain View, Calif.: Quintus Computer Systems, Inc., 1988c.

Roberts, G., *An Implementation of Prolog,* M.S. Thesis. Waterloo, Ontario: University of Waterloo, 1977.

Robinson, J., "A Machine-Oriented Logic Based on the Resolution Principle," *JACM,* Vol. 12, pp. 23–41. 1965.

Sandewall, E., "Conversion of Predicate-Calculus Axioms, Viewed as Non-Deterministic Programs," *International Joint Conference on Artificial Intelligence,* pp. 230–234. Stanford, Calif., 1973.

Schank, R. C., and R. P. Abelson, *Scripts, Plans, Goals, and Understanding.* Hillsdale, N.J.: Erlbaum, 1977.

Shaw, N., *A Formal System for Specifying and Verifying Program Performance,* Technical Report. Pittsburgh: Carnegie-Mellon University, 1980.

Shiel, B., "Power Tools for Programmers," *Datamation,* pp. 131–144, February 1983.

Simori, D. T., and S. H. Starr, "The Mission Oriented Approach to NATO C2 planning," *Signal,* September 1987.

Simmons, R. S., and J. Slocum, "Generating English Discourse from Semantic Networks," *CACM,* Vol. 15, pp. 891–905, 1972.

Steele, G. L., *Common LISP: The Language.* Burlington, Mass.: Digital Press, 1984.

Sterling, L., and E. Shapiro, *The Art of Prolog: Advanced Programming Techniques.* Cambridge, Mass.: The MIT Press, 1986.

SunView Applications Programmer's Guide. Mountain View, Calif.: Sun Microsystems, Inc., 1986.

Sweet, R., "An Evolving C2 Evaluation Tool—MCES Theory," *Proceedings of the 9th MIT/ONR Workshop on C3 Systems.* Cambridge, Mass.: Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, December 1986.

Thayse, A., *From Standard Logic to Logic Programming.* New York: John Wiley and Sons, 1988.

Unix Interface Overview. Mountain View, Calif.: Sun Microsystems, Inc., 1986.

van Emden, M., "McDermott on Prolog: A Rejoinder," *SICART Newsletter,* Vol. 73, pp. 19–20, 1980.

Warren, D. H. D., and F. Pereira, "Prolog—The Language and Its Implementation Compared with LISP," *SIGPLAN Notices,* Vol. 12, No. 8, August 1977.

Warren, D. H. D., "An Improved Prolog Implementation which Optimizes Tail Recursion," *Proceedings of Logic Programming Workshop,* pp. 1–11. Debrecen, Hungary, 1980.

————, "A View of the Fifth Generation and Its Impact," *Proceedings of Conference on Japan and the Fifth Generation.* New York: Pergamon, 1982.

————, Foreword to L. Sterling and E. Shapiro, *The Art of Prolog: Advanced Programming Techniques.* Cambridge, Mass.: The MIT Press, 1986.

Waterman, D. A., *A Guide to Expert Systems.* Reading, Mass.: Addison-Wesley, 1985.

Weiss, S. M., and C. A. Kulikowski, *Designing Expert Systems.* Totowa, N.J.: Rowman and Allanheld, 1984.

Woods, D D., and E. Hollnagel, "Mapping Cognitive Demands in Complex Problem-Solving Worlds." In B. R. Gaines and J. H. Boose (eds.), *Knowledge Acquisition for Knowledge-Based Systems.* New York: Harcourt Brace Jovanovich, 1988.

Yourdon, E., *Managing the Structured Techniques.* Englewood Cliffs, N.J.: Prentice-Hall, 1979.

Ziles, S. N., "Types, Algebra, and Modeling." In M. L. Brodie, J. Mylopoulos, and J. W. Schmidt (eds.), *On Conceptual Modeling.* New York: Springer-Verlag, 1984.

APPENDIX A
CMLP INSTANCE DIRECTORY

1. Class: *Characteristic*

Instance 'Characteristic'(1)
Element: C2
Name: Degraded Operations
Description: Degraded Operations:
    The capability of a C2 system to sustain losses in C2 and supporting elements and continue operating at a lower level of capability.

Instance 'Characteristic'(2)
Element: C2
Name: Mobile
Description: Mobility:
    The capability of the C2 element or its supporting elements to relocate in an effort to confuse attacking forces and to increase survivability.

Instance 'Characteristic'(3)
Element: C2
Name: NBC Hardened
Description: NBC Hardened:
    The capability of C2 and supporting elements to withstand the effects of Nuclear, Biological and Chemical attacks through passive measures and continue operating.

Instance 'Characteristic'(4)
Element: C2
Name: Physical Security
Description: Physical Security:
    The property of C2 and supporting elements to have security measures in place to prevent and counter sabotage and terrorist attacks.

Instance 'Characteristic'(5)
Element: C2
Name: Survivable Communications
Description: Survivable Communications:
    The capability of the C2 and supporting elements to maintain communication through periods of attack and continue operating.

Instance 'Characteristic'(6)
Element: C2
Name: Back-up Capability
Description: Back-up Capability:
    The property of a defensive system to have redundant elements such that as elements are lost other elements assume processing responsibility.

Instance 'Characteristic'(7)
Element: C2
Name: Secure Communications
Description: Secure Communications:
    The property of the C2 elements to maintain communication systems that are not jammable by counter measures nor able to be intercepted by enemy forces.

Instance 'Characteristic'(8)
Element: C2
Name: Secure Data Processing
Description: Secure Data Processing:
    The property of the system and the data processing equipment to prevent unauthorized access to the data and operations of the system.

Instance 'Characteristic'(9)

Element: Sensors
Name: Number
Description: Sensor Number:
    The number of sensors which interface to the C2 system and therefore require data processing by the C2 system

Instance 'Characteristic'(10)
Element: Sensors
Name: Location
Description: Sensor Location:
    The location of the sensors with respect to the proximity to the boundary of the defended area or to the expected threat

Instance 'Characteristic'(11)
Element: Sensors
Name: Type
Description: Sensor Type:
    The type of sensor refers to the band of the electro- magnetic spectrum in which it operates, for example infrared, radar or laser radar or an individual band designation such as L-band or S-band for radar.

Instance 'Characteristic'(12)
Element: Sensors
Name: Coverage Volume
Description: Sensor Coverage Volume:
    The volume through-out which the sensor has the capability to detect a target of a specified size or characteristic.

Instance 'Characteristic'(13)
Element: Sensors
Name: Sensitivity
Description: Sensor Sensitivity:
    The ability of the sensor to detect the minimum target at the maximum range.

Instance 'Characteristic'(14)
Element: Sensors
Name: Resolution
Description: Sensor Resolution:
    The capability of the sensor to distinguish between individual objects oriented such that they have the minimum separation distance between them or the ability to discriminate between decoys and true objects.

Instance 'Characteristic'(15)
Element: Sensors
Name: Survivability
Description: Sensor Survivability:
    The ability of the sensor to protect itself from threat objects whose mission is to neutralize the sensor.

Instance 'Characteristic'(16)
Element: Sensors
Name: Capacity
Description: Sensor Capacity:
    The maximum number of objects the sensor is capable of reporting simultaneously.

Instance 'Characteristic'(17)
Element: Sensors
Name: Scan Rate
Description: Sensor Scan Rate:
    The rate at which the sensor sweeps through its entire coverage volume.

Instance 'Characteristic'(18)

Element: Weapons
Name: Number
Description: Weapon Number:
  The number of weapons which interface to the C2 system and therefore require data processing by the C2 system

Instance 'Characteristic'(19)
  Element: Weapons
  Name: Location
  Description: Weapon Location:
    The location of the weapons with respect to the proximity to the boundary of the defended area or to the expected threat

Instance 'Characteristic'(20)
  Element: Weapons
  Name: Type
  Description: Weapon Type:
    The weapon type refers to the type of fusing (proximity, contact), the type of warhead (conventional, nuclear), or the type of sensor the weapon uses to home in on its target (radar, infrared).

Instance 'Characteristic'(21)
  Element: Weapons
  Name: Effective Volume
  Description: Weapon Effective Volume:
    The volume of space in which the weapon has the design probability of kill against a specified target.

Instance 'Characteristic'(22)
  Element: Weapons
  Name: Sensitivity
  Description: Weapon Sensitivity:
    The capability of the weapon's sensor to maintain lock-on to its intended target in a countermeasures environment.

Instance 'Characteristic'(23)
  Element: Weapons
  Name: Probability of Kill
  Description: Weapon Probability of Kill:
    The design probability of kill of the weapon against a specific target.

Instance 'Characteristic'(24)
  Element: Weapons
  Name: Speed
  Description: Weapon Maximum Acceleration/Speed:
    The maximum acceleration/speed of the weapon based on its designed performance characteristics.

Instance 'Characteristic'(25)
  Element: Weapons
  Name: Firing Rate
  Description: Weapon Firing Rate:
    The maximum rate at which the weapon can reload or refire.

Instance 'Characteristic'(26)
  Element: Threats
  Name: Number
  Description: Threat Number:
    The number of threats which the enemy has in inventory and are expected to be brought to bear against friendly forces.

Instance 'Characteristic'(27)
  Element: Threats
  Name: Location (Primary basing)
  Description: Threat Location:
    The location of the threats with respect to the proximity to the boundary of the defended area.

Instance 'Characteristic'(28)
  Element: Threats
  Name: Spatial Density
  Description: Threat Spatial Density:
    The proximity of threats to each other typically resulting in the inability of the sensor system to determine the actual number of attacking objects.

Instance 'Characteristic'(29)
  Element: Threats
  Name: Speed
  Description: Threat Maximum Acceleration/Speed:
    The maximum acceleration/speed of the threat based on received intelligence data.

Instance 'Characteristic'(30)
  Element: Threats
  Name: Detectability
  Description: Threat Detectability:
    The characteristics of the threat which determine its sensor signature.

Instance 'Characteristic'(31)
  Element: Threats
  Name: Maneuverability
  Description: Threat Maneuverability:
    The ability of the threat to maneuver within its performance envelope, usually expressed in terms of g forces.

Instance 'Characteristic'(32)
  Element: Threats
  Name: Weapon Avoidance
  Description: Threat Weapon Avoidance:
    The capability of the threat to use countermeasures or maneuverability to evade weapons.

Instance 'Characteristic'(33)
  Element: Threats
  Name: Identifiability
  Description: Threat Identifiability:
    The characteristics of the threat that permit the sensor system to distinguish it from other objects.

2. Class: *Function*

Instance 'Function'(1)
Function Group: Sensor Control
Name: Blanking Control
Description: Blanking Control:
    The ability to determine areas of high false target reports from individual sensors and reject data in those areas by sensor type.

Instance 'Function'(2)
Function Group: Sensor Control
Name: Radiation Management
Description: Radiation Management:
    The ability to control sensors radiating in a specific area at a given time, with respect to other sensors such that the same sensor is not radiating in the same sector every scan. The object of this function is to provide sensor coverage over the entire defensive area while confusing the enemy and increasing sensor survivability by randomly altering sensor areas of radiation.

Instance 'Function'(3)
Function Group: Sensor Control
Name: ECCM Control
Description: ECCM Control:
    The ability to select specific capabilities of individual sensors to provide improved threat visibility in a high ECM environment

Instance 'Function'(4)
Function Group: Sensor Control
Name: Sensor Tasking   ·
Description: Sensor Tasking:
    The ability to direct sensors to scan specific sectors to provide higher probabilities of detection on presumed threat corridors.

Instance 'Function'(5)
Function Group: Tracking
Name: Sensor Data Acceptance
Description: Sensor Data Acceptance:
    The process by which sensor data is received, tested for transmission errors, time tagged and placed in the appropriate buffer for further processing.

Instance 'Function'(6)
Function Group: Tracking
Name: Coordinate Transformation
Description: Coordinate Transformation:
    The process which translates target reports received from the sensors from the sensor coordinate plane to a common coordinate plane.

Instance 'Function'(7)
Function Group: Tracking
Name: Registration
Description: Registration:
    The process which applies an algorithm to the received sensor data to determine positional errors, due to translation errors and viewing angle, and correct the errors.

Instance 'Function'(8)
Function Group: Tracking
Name: Fusion/Correlation
Description: Fusion/Correlation:
    The process by which reports from multiple sensors on the same target are determined to be the same target and are then associated with an existing system track or cause the initiation of a system track.

Instance 'Function'(9)
Function Group: Tracking
Name: Track Update
Description: Track Update:
    The process by which a system track is updated in terms of position, velocity and heading based on data received from the systems sensors and correlated with the track.

Instance 'Function'(10)
Function Group: Tracking
Name: Ambiguity Resolution
Description: Ambiguity Resolution:
    The process which resolves which sensor reports are associated with tracks when the correlation process cannot conclusively determine the which tracks the reports should be correlated with. This situation occurs when tracks with similar headings cross or intersect each others path. Another situation which results in an inability to correlate data to tracks occurs when targets are flying in formation and sensors are able to detect more than one target.

Instance 'Function'(11)
Function Group: Tracking
Name: Track Initiation
Description: Track Initiation:
    The process by which uncorrelated data is analyzed through multiple sensor scans to determine if a new track should be created or the reports are false alarms.

Instance 'Function'(12)
Function Group: Tracking
Name: Kill Assessment
Description: Kill Assessment:
    The process by which the system determines if a weapon intercepted its target based on loss of sensor data from the target.

Instance 'Function'(13)
Function Group: Threat Evaluation
Name: Raid Composition
Description: Raid Composition:
    The process by which the type of threat which comprise the raid are determined through analysis of sensor data and intelligence information.

Instance 'Function'(14)
Function Group: Threat Evaluation
Name: Flight Characteristics
Description: Flight Characteristic Analysis:
    The process which attempts to associate potential flight paths of threats with objects tracked by the system based on expected routes of attack.

Instance 'Function'(15)
Function Group: Threat Evaluation
Name: Strength
Description: Raid Strength:
    The process which determines the actual number of threat objects based on fusion and correlation of sensor data and intelligence information. The number of objects can be determined accurately based on the fact that, at most each object should only yield one report per sensor.

Instance 'Function'(16)
Function Group: Threat Evaluation
Name: Priority Ranking

Description: Priority Ranking:

The process which determines which objects are the most threatening and should therefore be intercepted first.

Instance 'Function'(17)
Function Group: Threat Evaluation
Name: Enemy Order of Battle Maintenance
Description: Enemy Order of Battle Maintenance:

The process which attempts to determine the number and type of enemy resources remaining tin inventory based on intelligence information and knowledge of the threat objects detected and killed.

Instance 'Function'(18)
Function Group: Threat Assessment/Identification
Name: Flight Route Correlation
Description: Flight Route Correlation:

The process which associates tracks with previously stored flight plan information to determine the identity or classification of the object.

Instance 'Function'(19)
Function Group: Threat Assessment/Identification
Name: Route Deviation Alert
Description: Route Deviation Alert:

The process which monitors tracks on a known flight path and provides a warning indication should the track deviate from the flight path.

Instance 'Function'(20)
Function Group: Threat Assessment/Identification
Name: IFF/SIF Processing
Description: IFF/SIF Processing:

The process which correlates IFF/SIF data with specific tracks for positive identification purposes.

Instance 'Function'(21)
Function Group: Threat Assessment/Identification
Name: Geographic Determination
Description: Geographic Determination:

The process which assigns an identity to a track based on its geographic area of initial detection or flight route through a designated area.

Instance 'Function'(22)
Function Group: Threat Assessment/Identification
Name: Challenge Processsing
Description: Challenge Processing:

The process which attempts to identify unknown tracks by requesting transmission of a code and receiving the appropriate coded reply.

Instance 'Function'(23)
Function Group: Threat Assessment/Identification
Name: Discrimination
Description: Discrimination Analysis:

The process which utilizes target signature data from the sensors and compares the data with stored information to attempt to determine the exact type of target.

Instance 'Function'(24)
Function Group: Weapon Assignment
Name: Weapon Status
Description: Weapon Status Maintenance:

The process which checks and maintains database comprising the operational and ready status of the weapons.

Instance 'Function'(25)
Function Group: Weapon Assignment

Name: Weapon Probability of Kill
Description: Weapon Probability of Kill:

The process which attempts to determine the weapon that has the highest probability of kill against a specific target.

Instance 'Function'(26)
Function Group: Weapon Assignment
Name: Weapon Intercept Time
Description: Weapon Intercept Processing:

The process which attempts to determine the time and location at the point where the weapon will intercept the threat and the relative position of the threat with respect to the defensive perimeter.

Instance 'Function'(27)
Function Group: Weapon Assignment
Name: Weapon-Target Assignment
Description: Weapon Target Assignment:

The process which determines the weapons that should be committed against a particular threat based on priority ranking, weapon probability of kill, and weapon intercept time.

Instance 'Function'(28)
Function Group: Weapon Control
Name: Weapon Solution Generation
Description: Intercept Solution Generation:

The process which computes and recomputes the weapon intercept paths to update the weapons to provide the best intercept path within weapon performance limits.

Instance 'Function'(29)
Function Group: Weapon Control
Name: Weapon Guidance
Description: Interceptor Guidance:

The process which provides detailed flight route information, including altitude, heading, speed and course updates, through pre-defined tactics, to guide an interceptor to the point where it will intercept its target.

Instance 'Function'(30)
Function Group: Weapon Control
Name: Target Update Processing
Description: Target Update Processing:

The process which updates the threat location and predicts the future location so that appropriate direction can be given to the weapons.

Instance 'Function'(31)
Function Group: Weapon Control
Name: Weapon Effectiveness
Description: Weapon Effectiveness:

The process which maintains the score of weapons that successfully intercepted their intended targets and the total weapons committed against targets. Additionally, monitors the missions for shoot-look-shoot mission planning.

Instance 'Function'(32)
Function Group: Telling
Name: Data Receipt
Description: Data Receipt:

The process which accepts data from supporting and superordinate nodes, verifies the message integrity and routes the message to the addressee.

Instance 'Function'(33)
Function Group: Telling
Name: Position Translation
Description: Position Translation:

The process which performs the translation from the

coordinate plane of the reporting node to the local coordinate plane based on the Data Link Reference Position(DLRP).

Instance 'Function'(34)
    Function Group: Telling
    Name: Data Transmission
    Description: Data Transmission:
        The process which collects system data from all system functions and routes it for transmission to the supporting nodes.

Instance 'Function'(35)
    Function Group: Telling
    Name: Input/Output Filtering (GEO,TYPE,ID)
    Description: Input/Output Filtering:
        The process which selects the data to be transmitted or received based on geography, type or identification.

Instance 'Function'(36)
    Function Group: Telling
    Name: Reporting Responsibility
    Description: Reporting Responsibility:
        The process which determines which of the nodes reporting a track has the ultimate responsibility for the tracks information. Usually based on track data quality and is arbitrated on an individual link basis for those nodes detecting the track.

Instance 'Function'(37)
    Function Group: Telling
    Name: Link Status Reporting
    Description: Link Status Monitoring:
        The process which maintains a message count for each link and reports total messages, messages received in error, messages re-transmitted, current availability of the link and the messages per transmission frame.

Instance 'Function'(38)
    Function Group: Telling
    Name: Alerts/Warning Info
    Description: Alerts/Warning Information:
        The process which transmits alert/warning data to other nodes upon detection of hostile activity or imminent attack.

Instance 'Function'(39)
    Function Group: Telling
    Name: Authority Control Arbitration
    Description: Authority Control Arbitration:
        The process which determines which nodes are subordinate or super-ordinate as nodes are lost to hostile forces.

Instance 'Function'(40)
    Function Group: Executive (OS)
    Name: Real Time Control
    Description: Real Time Control:
        The process which monitors the amount of data the system is processing, provides alerts as system capacity is approached, expands and contracts the frame, as necessary, or drops excess data as it is received to prevent catastrophic system failures.

Instance 'Function'(41)
    Function Group: Executive (OS)
    Name: System Monitoring/Recovery
    Description: System Monitoring/Recovery:
        The process which statuses system hardware and software to ensure system integrity, and performs required actions to re-allocate and reconfigure the system to maintain processing capability and provide alerts to the operator in the event of a malfunction.

Instance 'Function'(42)

Function Group: Executive (OS)
Name: Recording
Description: System Recording:
    The process which records selected live or simulated data for offline processing, training and procedure review.

Instance 'Function'(43)
    Function Group: Executive (OS)
    Name: Simulation
    Description: System Simulation:
        The process which provides the capability to simulate system inputs for the purpose of system test, exercise, and training.

Instance 'Function'(44)
    Function Group: Executive (OS)
    Name: Operator Input Processing
    Description: Operator Input Processing:
        The process which responds to operator requests for information for any item in the system or any action taken by the system.

Instance 'Function'(45)
    Function Group: Executive (OS)
    Name: Display Generation
    Description: Display Generation:
        The process which formats data for display to the operators and routes the data to the appropriate display devices.

3. Class: *Function Group*

Instance 'Function Group'(1)
Name: Sensor Control
Description: Sensor Control Function Group:
   The Sensor Control Function Group contains the functions which the C2 system performs to maintain complete sensor coverage over the entire protected region and its approaches and prevents hostile forces from deceiving the sensor systems.
Function: [Blanking Control,Radiation Management,ECCM Control,Sensor Tasking]

Instance 'Function Group'(2)
Name: Tracking
Description: Tracking Function Group:
   The Tracking Function Group contains the functions that are responsible for receiving and processing raw sensor data for presentation to the operator and for use by other system functions.
Function: [Sensor Data Acceptance,Coordinate Transformation,Registration,Fusion/Correlation,Track Update,Ambiguity Resolution,Track Initiation,Kill Assessment]

Instance 'Function Group'(3)
Name: Threat Evaluation
Description: Threat Evaluation Function Group:
   The Threat Evaluation Function Group contains the functions that attempt to determine the level of threat represented by objects detected and tracked by the system.
Function: [Raid Composition,Flight Characteristics,Strength,Priority Ranking,Enemy Order of Battle Maintenance]

Instance 'Function Group'(4)
Name: Threat Assessment/Identification
Description: Threat Assessment and Identification Function Group:
   The Threat Assessment and Identification Function Group contains the functions which are responsible for determining the identity of an object and assessing the threat imposed by the object.
Function: [Flight Route Correlation,Route Deviation Alert,IFF/SIF Processing,Geographic Determination,Challenge Processssing,Discrimination]

Instance 'Function Group'(5)
Name: Weapon Assignment
Description: Weapon Assignment Function Group:
   The Weapon Assignment Function Group contains the functions responsible for the "pairing" of specific weapon to its intended target.
Function: [Weapon Status,Weapon Probability of Kill,Weapon Intercept Time,Weapon-Target Assignment]

Instance 'Function Group'(6)
Name: Weapon Control
Description: Weapon Control Function Group:
   The Weapon Control Function Group contains the functions responsible for computing the guidance information necessary for the weapon to reach its intended target.
Function: [Weapon Solution Generation,Weapon Guidance,Target Update Processing,Weapon Effectiveness]

Instance 'Function Group'(7)
Name: Telling
Description: Telling Function Group:
   The Telling Function Group contains the functions responsible for maintain data communications between the C2 system and its supporting elements.
Function: [Data Receipt,Position Translation,Data Transmission,Input/Output Filtering (GEO,TYPE,ID),Reporting Responsibility,Link Status Reporting,Alerts/Warning Info,Authority Control Arbitration]

Instance 'Function Group'(8)
Name: Executive (OS)
Description: Executive Functions Group:
   The Executive Functions Group contains the functions responsible for the basic system operations including hardware and software error detection.
Function: [Real Time Control,System Monitoring/Recovery,Recording,Simulation,Operator Input Processing,Display Generation]

Instance 'Function Group'(1)
Name: Sensor Control
Description: Description
Function: [Blanking Control,Radiation Management,ECCM Control,Sensor Tasking]

Instance 'Function Group'(2)
Name: Tracking
Description: Description
Function: [Sensor Data Acceptance,Coordinate Transformation,Registration,Fusion/Correlation,Track Update,Ambiguity Resolution,Track Initiation,Kill Assessment]

Instance 'Function Group'(3)
Name: Threat Evaluation
Description: Description
Function: [Raid Composition,Flight Characteristics,Strength,Priority Ranking,Enemy Order of Battle Maintenance]

Instance 'Function Group'(4)
Name: Threat Assessment/Identification
Description: Description
Function: [Flight Route Correlation,Route Deviation Alert,IFF/SIF Processing,Geographic Determination,Challenge Processssing,Discrimination]

Instance 'Function Group'(5)
Name: Weapon Assignment
Description: Description
Function: [Weapon Status,Weapon Probability of Kill,Weapon Intercept Time,Weapon-Target Assignment]

Instance 'Function Group'(6)
Name: Weapon Control
Description: Description
Function: [Weapon Solution Generation,Weapon Guidance,Target Update Processing,Weapon Effectiveness]

Instance 'Function Group'(7)
Name: Telling
Description: Description
Function: [Data Receipt,Position Translation,Data Transmission,Input/Output Filtering (GEO,TYPE,ID),Reporting Responsibility,Link Status Reporting,Alerts/Warning Info,Authority Control Arbitration]

Instance 'Function Group'(8)
Name: Executive (OS)
Description: Description
Function: [Real Time Control,System Monitoring/Recovery,Recording,Simulation,Operator Input Processing,Display Generation]

4. Class: *Goal*

Instance 'Goal'(1)
   Name: Maximum Case Attack
   Description: The C2 System should be capable of accommodating the maximum case attack.
   Goal Group: System Robustness
   Satisfactory: SATISFACTORY - The C2 System can accommodate the maximum expected simultaneous attack when in conformance with all expected scenarios.
   Partially Satisfactory: PARTIALLY SATISFACTORY - The C2 System can accommodate the maximum attack only if the threat is disbursed and uses multiple attack corridors. The C2 System may be designed to operate in a degraded mode when faced with maximum threat in corridors.
   Unsatisfactory: UNSATISFACTORY - The C2 System will saturate when the maximum threat attack occurs.
   Not Supported: NOT SUPPORTED - This goal is not supported when the sensor systems cannot detect all of the attack or the communication system cannot report all of the attack. This rating may be present with other ratings indicating what the C2 System would do if the remainder of the system could support the maximum case attack.
   Not Applicable: NOT APPLICABLE - This goal is expected to apply to all C2 Systems.
   Applicable Functions: None
   Affects Swat: No
   Affects Capacity: Yes
   Affects Geography: No
   Additional Remarks:

Instance 'Goal'(2)
   Name: Processing for all Regions
   Description: The C2 System should provide processing capability to accommodate all geographic areas.
   Goal Group: System Robustness
   Satisfactory: SATISFACTORY - The C2 System can withstand the maximum attack in any geographic area without saturation but may require degraded performance.
   Partially Satisfactory: PARTIALLY SATISFACTORY - The system may saturate in one or more geographic areas when subjected to the maximum attack or there are areas in which there is no command and control support, but the system is expected to operate within goals.
   Unsatisfactory: UNSATISFACTORY - The system cannot meet the threat environment in expected areas of attack.
   Not Supported: NOT SUPPORTED - This goal is not supported when the sensor (or communications) systems do not provide the capability necessary to detect and report the threat in any of the geographic areas. This rating may occur with one of the ratings above which is interpreted as to how the C2 would behave if the capability was supported by other elements.
   Not Applicable: NOT APPLICABLE - This rule is not applicable when System is not broken down into geographic regions.
   Applicable Functions: [Sensor Tasking]
   Affects Swat: No
   Affects Capacity: Yes
   Affects Geography: Yes
   Additional Remarks:

Instance 'Goal'(3)
   Name: Conserve Defensive Resources
   Description: The system should conserve defensive resources for protection in future attacks based upon current attack strengths, enemy inventories, and assets being defended.
   Goal Group: System Robustness

   Satisfactory: SATISFACTORY - The system provides for automatic or manual aids to analyze the current attack, enemy inventories and status of the defended assets to determine the level of engagement of the current attack.
   Partially Satisfactory: PARTIALLY SATISFACTORY - The system provides sufficient information to man so that he may perform an informed, but unassisted, decision on the extent of the defense information on enemy inventories of threat so that the system cannot make a informed decision on the level of defense.
   Unsatisfactory: UNSATISFACTORY - The system expends resources at will, or provides conditions under which man is likely to expend resources at will, even when that is not in the best interest in meeting the goals of the defensive system.
   Not Supported: NOT SUPPORTED - There is insufficient attack information, status of defended assets, or information on enemy inventories of threat such that the system cannot make an informed decision on the level of defense.
   Not Applicable: NOT APPLICABLE - The defensive system has been designed to use its maximum capability for any attack.
   Applicable Functions: [Raid Composition,Strength,Priority Ranking,Enemy Order of Battle Maintenance,Weapon Probability of Kill]
   Affects Swat: Yes
   Affects Capacity: No
   Affects Geography: No
   Additional Remarks:

Instance 'Goal'(4)
   Name: Expend Resources Proportionally
   Description: The C2 System shall expend defensive resources proportional to the value of assets under attack.
   Goal Group: System Robustness
   Satisfactory: SATISFACTORY - The system provides for automatic or manual aids to analyze the assets under attack to determine appropriate level of engagement.
   Partially Satisfactory: PARTIALLY SATISFACTORY - The system provides sufficient information to man so that he may perform an informed, but unassisted, decision on the extent of the defense
   Unsatisfactory: UNSATISFACTORY - The system expends resources at will, or provides conditions under which man is likely to expend resources at will, even when that is not in the best interest in meeting the goals of the defensive system.
   Not Supported: NOT SUPPORTED - There is insufficient attack information, so that the system cannot make a informed decision on the level of defense.
   Not Applicable: NOT APPLICABLE - This system does not have the capability to identify the assets being threatened and must perform a subtractive defense.
   Applicable Functions: None
   Affects Swat: No
   Affects Capacity: No
   Affects Geography: No
   Additional Remarks:

Instance 'Goal'(5)
   Name: Use all Data Received
   Description: The C2 System should utilize all received surveillance data to the greatest extent possible.
   Goal Group: Surveillance Data Use
   Satisfactory: SATISFACTORY - The C2 System will accept and analyze all received surveillance data in sufficient quantities to have appropriate information for all missions.
   Partially Satisfactory: PARTIALLY SATISFACTORY - The C2 System uses sufficient data to perform primary missions with some confidence.
   Unsatisfactory: UNSATISFACTORY - The C2 System fails

to use data and makes uniformed decisions at times.

Not Supported: NOT SUPPORTED - *Sensors (or communications) do not provide all data or information from data.*

Not Applicable: NOT APPLICABLE - There is no need for using more sensor data than currently being used.

Applicable Functions: [Fusion/Correlation,Track Update,Ambiguity Resolution,Track Initiation,Kill Assessment,Strength,Flight Route Correlation,Route Deviation Alert,IFF/SIF Processing,Geographic Determination,Discrimination]

Affects Swat: No

Affects Capacity: No

Affects Geography: No

Additional Remarks:

Instance 'Goal'(6)

Name: Not Rely on Deniable Data

Description: The C2 System shall not rely upon the use of deniable intelligence data for any processing, but shall provide the capability to use the data to the best advantage upon receipt.

Goal Group: Surveillance Data Use

Satisfactory: SATISFACTORY - The C2 System accepts intelligence data and formulates this data into order of battle information that is used for threat assessment.

Partially Satisfactory: PARTIALLY SATISFACTORY - The C2 System allows the command and control operator to input rules of engagement based upon manual interpretation of intelligence data.

Unsatisfactory: UNSATISFACTORY - This system makes no use of intelligence data following establishment of alert status, or the system depends upon intelligence data that is deniable by the enemy.

Not Supported: NOT SUPPORTED - The intelligence system does not provide usable data.

Not Applicable: NOT APPLICABLE - No intelligence data exists that can be made available.

Applicable Functions: None

Affects Swat: No

Affects Capacity: No

Affects Geography: No

Additional Remarks:

Instance 'Goal'(7)

Name: Correlate & Fuse Data

Description: The C2 System shall correlate and fuse data to obtain estimates of raid counts and attack character.

Goal Group: Surveillance Data Use

Satisfactory: SATISFACTORY - C2 System provides capability to use any threat data that is available and provides raid size counting and analysis.

Partially Satisfactory: PARTIALLY SATISFACTORY - The C2 System provides a capability to estimate overall raid counts without any to use all data to determine the nature of the attack.

Unsatisfactory: UNSATISFACTORY - The C2 System responds to detected targets individually without attempting to analyze the nature of the attack.

Not Supported: NOT SUPPORTED - The sensors (or communications) do not provide enough data to do any significant threat assessment.

Not Applicable: NOT APPLICABLE - There is no need for the C2 System to analyze overall attack carriers as all assignments should be made on an individual threat basis.

Applicable Functions: [Fusion/Correlation,Raid Composition,Priority Ranking,Target Update Processing]

Affects Swat: No

Affects Capacity: No

Affects Geography: No

Additional Remarks:

Instance 'Goal'(8)

Name: Support Engagements

Description: The C2 System shall fuse (not merge) sensor data to obtain the best possible identification and position estimates to support engagement.

Goal Group: Surveillance Data Use

Satisfactory: SATISFACTORY - The C2 System performs positive correlation and fuses multi-sensor data.

Partially Satisfactory: PARTIALLY SATISFACTORY - The C2 System fuses data when there are single correlation errors but does not include complex correlated algorithms to select the best sensor data to perform a more accurate correlation.

Unsatisfactory: UNSATISFACTORY - Data from single sensor is used at different times of flight.

Not Supported: Not SATISFACTORY - Sensors (or communications) do not provide sufficient data for multi-sensor communications

Not Applicable: NOT APPLICABLE - For the C2 System, single sensor data exceeds accuracy requirements.

Applicable Functions: [Fusion/Correlation]

Affects Swat: No

Affects Capacity: No

Affects Geography: No

Additional Remarks:

Instance 'Goal'(9)

Name: Control Resources

Description: The C2 System shall utilize surveillance data to track and control defense system resources and monitor compliance with direction.

Goal Group: Surveillance Data Use

Satisfactory: SATISFACTORY - Will accept and analyze surveillance data in sufficient time to take any available appropriate action for all engagements.

Partially Satisfactory: PARTIALLY SATISFACTORY - The C2 System will accept and analyze surveillance data when available, will support only some of the alternative actions, or will support only some of the engagements.

Unsatisfactory: UNSATISFACTORY - Fails to use available action in more than 25% of unsuccessful engagements.

Not Supported: NOT SUPPORTED - Sensors (or communications) do not provide needed data.

Not Applicable: NOT APPLICABLE - There are no misses caused by lack of control and/or there are not alternative actions in this system.

Applicable Functions: [Track Update]

Affects Swat: No

Affects Capacity: No

Affects Geography: No

Additional Remarks:

Instance 'Goal'(10)

Name: Confirm/Verify Engagement Results

Description: The C2 System should utilize surveillance data to confirm/verify engagement results.

Goal Group: Surveillance Data Use

Satisfactory: SATISFACTORY - The system provides the capability to perform automatic kill assessment by infusing weapon feedback with surveillance data.

Partially Satisfactory: PARTIALLY SATISFACTORY - The system provides the capability to accept weapon feedback with surveillance data and allows the operator to use this data for kill assessment.

Unsatisfactory: UNSATISFACTORY - The system does not use surveillance data from kill assessment.

Not Supported: NOT SUPPORTED by System - The sensors (or communications) do not provide data that is usable to

support kill assessment.
Not Applicable: NOT APPLICABLE - The system does not have time to utilize kill assessment data.
Applicable Functions: [Kill Assessment]
Affects Swat: No
Affects Capacity: No
Affects Geography: No
Additional Remarks:

Instance 'Goal'(11)
Name: Consider Assets and Risk
Description: The C2 System shall consider the value of assets at risk, degree of risk, probability of defensive actions success, and the risk to assets that may result from defensive actions.
Goal Group: System Response Selection
Satisfactory: SATISFACTORY - The system includes threat evaluation and priority ranking and utilizes weapon threat-engagement probabilities to select a defensive action.
Partially Satisfactory: PARTIALLY SATISFACTORY - The system utilizes limited information and rules of thumb for assigning weapons to specific threats or makes unsupported assumptions in the nature of the threat type for performing assignment.
Unsatisfactory: UNSATISFACTORY - The system responds to the existence of a threat to assign arbitrary weapons without considering the probability of engagement success and need for engagement.
Not Supported: NOT SUPPORTED - The sensors (or communications) do not provide threat assessment information or there is no data available on the performance of our weapons on the enemies threat.
Not Applicable: NOT APPLICABLE - There is only one kind of threat and one kind of weapon (or all threat/weapon engagements are "?equally likely) and all targets should be engaged.
Applicable Functions: [Raid Composition,Strength,Priority Ranking,Weapon Probability of Kill,Weapon-Target Assignment]
Affects Swat: Yes
Affects Capacity: No
Affects Geography: No
Additional Remarks:

Instance 'Goal'(12)
Name: Optimize Time to Intercept
Description: The C2 should attempt to optimize the time to intercept to achieve the most efficient weapon usage for the value of assets at risk.
Goal Group: System Response Selection
Satisfactory: SATISFACTORY - The C2 System provide the capability to identify threat parameters and assigns weapons for optimal engagement (or to allow shoot-look-shoot capabilities as is appropriate).
Partially Satisfactory: PARTIALLY SATISFACTORY - The system provides the capability to perform weapon assignment but does not provide the weapon with sufficient information to optimize the engagement.
Unsatisfactory: UNSATISFACTORY - The system assigns weapons to targets but does not provide data or control to achieve the highest possible engagement result.
Not Supported: NOT SUPPORTED - The sensors (or communications) do not provide sufficient threat data to allow optimization of engagements.
Not Applicable: NOT APPLICABLE - In this defensive system there is no reason to optimize engagements as all engagements are equally likely and shoot-look-shoot is not desirable.
Applicable Functions: [Weapon Probability of Kill,Weapon Intercept Time,Weapon-Target Assignment]
Affects Swat: Yes
Affects Capacity: No

Affects Geography: No
Additional Remarks:

Instance 'Goal'(13)
Name: Multiple Weapons Commitments
Description: The C2 System shall be capable of supporting shoot-look-shoot and shoot-fail-shoot weapon commitments against a single target to ensure effectiveness.
Goal Group: System Response Selection
Satisfactory: SATISFACTORY - The system will analyze data and determine the desirability to take early engagements to allow sufficient time for shoot- look-shoot and shoot-fail-shoot. The system will provide sufficient kill assessment and weapon control to exercise shoot-look-shoot or shoot- fail-shoot.
Partially Satisfactory: PARTIALLY SATISFACTORY - The system will take best first shot engagements and will shoot-look- shoot and shoot-fail-shoot if there is sufficient time.
Unsatisfactory: UNSATISFACTORY - The system does not support shoot-look-shoot and shoot-fail-shoot type decisions either because it is not structured to consider early engagements or because it doesn't have the capability to perform kill assessment and weapon assignment.
Not Supported: NOT SUPPORTED - The sensors (or communications) do not provide sufficiently early information to allow shoot-look-shoot, or shoot-fail-shoot or do not provide kill assessment allowing alternative weapons to be assigned, or this system has insufficient weapons to support shoot-look-shoot or shoot-fail-shoot.
Not Applicable: NOT APPLICABLE - The engagement sequence for this defensive system does not allow sufficient time for shoot-look-shoot and shoot-fail-shoot.
Applicable Functions: [Weapon Status,Weapon Intercept Time,Weapon-Target Assignment]
Affects Swat: No
Affects Capacity: No
Affects Geography: No
Additional Remarks:

Instance 'Goal'(14)
Name: Environmental & Geometric Conditions
Description: The C2 System shall consider known doctrinal employment, intelligence, environmental and geometric conditions to select a weapon that has the highest probability of success in engagement approach.
Goal Group: System Response Selection
Satisfactory: SATISFACTORY - The C2 System provides automatic (or assistance to operator) analysis of engagement geometries to allow selection of the highest probability engagement sequence.
Partially Satisfactory: PARTIALLY SATISFACTORY - The C2 System considers the most important factors leading to engagement success but ignores functions such as sun angle that may lead to infrequent nonoptimized engagement.
Unsatisfactory: UNSATISFACTORY - The system does not analyze available data in order to select high probability engagements and sufficient information is not provided to the weapons system to support engagement planning.
Not Supported: NOT SUPPORTED - The sensors (or communications) do not provide sufficient information to allow the C2 or weapon systems to select higher probability engagements.
Not Applicable: NOT APPLICABLE - The engagement sequence is not subject to doctrinal employment, environmental, or geometric conditions to determine its success.
Applicable Functions: None
Affects Swat: No
Affects Capacity: No
Affects Geography: No
Additional Remarks:

Instance 'Goal'(15)

Name: Give Updates to Weapons

Description: The C2 System shall provide weapons with updates of threat status and position as required by the weapons system.

Goal Group: System Response Selection

Satisfactory: SATISFACTORY - The C2 System continues to accept and analyze surveillance data during the engagement sequence and provides any data to the weapons that will facilitate the engagement.

Partially Satisfactory: PARTIALLY SATISFACTORY - The C2 System accepts some of the data that is available from the surveillance system and relays that on to the weapons, but does not provide all data available that could support a satisfactory engagement.

Unsatisfactory: UNSATISFACTORY - The C2 System does not include provisions for passing data on to the weapons systems.

Not Supported: NOT SUPPORTED - The sensors (or communications) do not provide the data that might be useful to the weapon during the engagement sequence.

Not Applicable: NOT APPLICABLE - The weapons system provides its own sensor data in sufficient quantities or does not need further target data so it does not desire any further surveillance data once engagement sequence is initiated.

Applicable Functions: [Target Update Processing]

Affects Swat: No

Affects Capacity: No

Affects Geography: No

Additional Remarks:

Instance 'Goal'(16)

Name: Send Data & Directives in Sufficient Time

Description: The C2 System shall identify threats, calculate threat parameters, and disperse data and force directives to command authorities forces and other agencies in sufficient time to prepare and execute defensive measures.

Goal Group: System Timeliness

Satisfactory: SATISFACTORY - The C2 System provides sufficient threat warning and threat data to control defensive engagements, and allow friendly assets to perform defensive measures.

Partially Satisfactory: PARTIALLY UNSATISFACTORY - The C2 System provides data to support some potential countermeasures to threat but does not provide threat data in sufficient time to support all varieties of passive countermeasures and alternatives of threat interdiction.

Unsatisfactory: UNSATISFACTORY - The C2 System does not provide data sufficiently early to support defense of the assets under attack.

Not Supported: NOT SUPPORTED - The sensor systems (or communications) are not designed to provide to the C2 System sufficient data that could be available to support selected defense activities.

Not Applicable: NOT APPLICABLE - Either the threat data does not exist sufficiently early to allow defense system action timeliness is not an issue for this system.

Applicable Functions: [Flight Route Correlation,IFF/SIF Processing,Geographic Determination,Challenge Processsing,Discrimination,Alerts/Warning Info]

Affects Swat: No

Affects Capacity: No

Affects Geography: No

Additional Remarks:

Instance 'Goal'(17)

Name: Timely Recovery From Failure

Description: The C2 System should provide for timely recovery in the event of system failure such that system operations are not degraded.

Goal Group: System Timeliness

Satisfactory: SATISFACTORY - The C2 System includes sufficient fast error detection, analysis, and recovery logic such that the system can continue normal operation when faced with single failures or any multiple failures that exceed a probability of .01 of occurrence.

Partially Satisfactory: PARTIALLY SATISFACTORY - The system will recover from most failures in sufficient time to prevent command and control service disruption.

Unsatisfactory: UNSATISFACTORY - There exists a probability of higher than .01 that the system will fail to accomplish its mission because of the error recovery time.

Not Supported: NOT SUPPORTED - This goal is concerned with failures within the C2 System as well as failures within the weapons and sensors. It is not supported in terms of recovery from sensor, weapon or communication failures, if these systems have not been designed to provide sufficient error data or backup capability to allow failure recovery.

Not Applicable: NOT APPLICABLE - This condition is believed to be applicable for all C2 Systems.

Applicable Functions: [Real Time Control,System Monitoring/Recovery]

Affects Swat: No

Affects Capacity: No

Affects Geography: No

Additional Remarks:

Instance 'Goal'(18)

Name: Process Data Fast Enough for Users

Description: The C2 System shall provide response capability such that the responses are not delayed upon receipt of the maximum amount of data from external elements.

Goal Group: System Timeliness

Satisfactory: SATISFACTORY - The C2 System provides sufficient bandwidths in all of its functions so they can handle the maximum amount of data expected and still meet mission timelines.

Partially Satisfactory: PARTIALLY SATISFACTORY - The C2 System may overload under very extreme conditions causing some (degradation of system performance but not complete loss of mission.

Unsatisfactory: UNSATISFACTORY - The C2 System may become saturated and fail to provide the necessary control to accomplish a mission.

Not Supported: NOT SUPPORTED - The sensors, communications or weapons will easily saturate in this system so that the C2 System cannot accomplish the mission when confronted with selected threats scenarios.

Not Applicable: NOT APPLICABLE - The threats that this system is concerned with are not sufficiently numerous as to be able to saturate the C2 System.

Applicable Functions: [Real Time Control]

Affects Swat: No

Affects Capacity: Yes

Affects Geography: No

Additional Remarks:

Instance 'Goal'(19)

Name: Defined Authority Chain

Description: The C2 System shall be based upon a defined authoritative chain to ensure positive command and control in all situations (war and peace).

Goal Group: System Authority Control

Satisfactory: SATISFACTORY - The C2 System provides survivable capability to support each necessary level of the command hierarchy and allow them to accomplish their decision functions.

Partially Satisfactory: PARTIALLY SATISFACTORY - The

C2 System provides hierarchical control that must reassign authority under certain conditions to be able to meet the mission goals.
Unsatisfactory: UNSATISFACTORY - The C2 System does not survive in a variety of credible attacks or lacks connectivity across the authority chain.
Not Supported: NOT SUPPORTED - This goal is not supported when there is insufficient communications within the system to allow implementation of the defined authority chain.
Not Applicable: NOT APPLICABLE - This goal is believed to be applicable to all missions.
Applicable Functions: [Authority Control Arbitration]
Affects Swat: No
Affects Capacity: No
Affects Geography: No
Additional Remarks:

Instance 'Goal'(20)
Name: Collect Data on System Performance
Description: The C2 System shall maintain data on system performance and report that data to appropriate authority upon request.
Goal Group: System Authority Control
Satisfactory: SATISFACTORY - The C2 System has a capability to perform automatic recording of system performance and feedback to higher authority.
Partially Satisfactory: PARTIALLY SATISFACTORY - The C2 System has a capability to record performance data but requires manual analysis of that data.
Unsatisfactory: UNSATISFACTORY - The system does not have the capability to monitor its own performance or to record data associated with that performance.
Not Supported: NOT SUPPORTED - This goal should be supported within the C2 System.
Not Applicable: NOT APPLICABLE - This goal was believed to be appropriate for all command and control systems.
Applicable Functions: [System Monitoring/Recovery,Recording]
Affects Swat: No
Affects Capacity: Yes
Affects Geography: No
Additional Remarks:

Instance 'Goal'(21)
Name: Decisions Based on Roe
Description: The C2 System shall base decisions and control processes on rules of engagement which allow manual control to be maintained without manual stress to accomplish a process.
Goal Group: System Authority Control
Satisfactory: SATISFACTORY - The C2 System is designed in such a way that man can control the activities of the system and is provided with sufficient data to make command level decisions.
Partially Satisfactory: PARTIALLY SATISFACTORY - Under certain stress conditions, the operators of the system are provided with more data than they can handle or are required to make detailed decisions before action that exceeds their ability to respond.
Unsatisfactory: UNSATISFACTORY - Either the C2 System will take control away from the man or will not perform the basic mission of the system without overloading the man with decision responsibility and/or data.
Not Supported: NOT SUPPORTED - This goal can be supported within the C2 System.
Not Applicable: NOT APPLICABLE - This goal is applicable to all C2 Systems
Applicable Functions: [Track Update,Route Deviation Alert,Weapon-Target Assignment,Enemy Order of Battle

Maintenance,Weapon Solution Generation]
Affects Swat: No
Affects Capacity: No
Affects Geography: No
Additional Remarks:

Instance 'Goal'(22)
Name: All Anticipated Environments
Description: The C2 System shall be operable in all anticipated environments.
Goal Group: System Survivability
Satisfactory: SATISFACTORY - The C2 System is designed to be robust to all anticipated attacks upon command and control and can provide sufficient control to allow the system to operate.
Partially Satisfactory: PARTIALLY SATISFACTORY - The C2 will meet most threat environments, but may be forced into graceful degradation as C2 System losses are sustained under particular attack scenarios.
Unsatisfactory: UNSATISFACTORY - The C2 Systems shall catastrophically fail when the enemy elects to attack the C2 System.
Not Supported: NOT SUPPORTED - This goal can be supported within the C2 System.
Not Applicable: NOT APPLICABLE - This goal is applicable to all C2 Systems
Applicable Functions: None
Affects Swat: No
Affects Capacity: No
Affects Geography: No
Additional Remarks: [Degraded Operations,Mobile,NBC Hardened]

Instance 'Goal'(23)
Name: Graceful Degradation
Description: The C2 System should be capable of operation in reduced performance modes in event of loss of sensors, weapons, communications or other assets. The system shall gracefully degrade as system losses are sustained.
Goal Group: System Survivability
Satisfactory: SATISFACTORY - The C2 System is designed for degraded operation as sensor data, weapons, or communications is lost. The remaining elements within the system will be used to the maximum extent to accomplish the mission.
Partially Satisfactory: PARTIALLY SATISFACTORY - The C2 System is designed to be able to cope with loss of system assets. Under some conditions non-optimal responses to the threat will be taken because of inappropriate C2 action.
Unsatisfactory: UNSATISFACTORY - The C2 System will breakdown under classes of attack on systems assets and fail to obtain appropriate use of the surviving assets.
Not Supported: NOT SUPPORTED - This goal is not supported if critical elements, such as a communication system, can cause disruption of command and control activity.
Not Applicable: NOT APPLICABLE - This goal is applicable to all C2 Systems.
Applicable Functions: None
Affects Swat: No
Affects Capacity: Yes
Affects Geography: No
Additional Remarks: [Degraded Operations,Back-up Capability]

Instance 'Goal'(24)
Name: Balanced Decision Making
Description: The C2 System shall provide a balance of automated and manual decision making such that operators are not overloaded with information or decisions.
Goal Group: Amount of Manual Participation

Satisfactory: SATISFACTORY - The C2 System is designed to assist the operator in making his decisions and allows the operator manual control when so selected by the operator.
Partially Satisfactory: PARTIALLY SATISFACTORY - Under extreme attack scenarios, the C2 System may provide too much information to the operator, or require more control than he can provide causing a slight degradation in system performance.
Unsatisfactory: UNSATISFACTORY - The C2 System performs all decisions automatically without manual override or saturates the operator with more data and decision responsibility than the operator can handle in likely attack modes.
Not Supported: NOT SUPPORTED - This goal is not supported by a system when there is insufficient time to allow manual decision processes so that all processes must be automated.
Not Applicable: NOT APPLICABLE - This goal is applicable to C2 Systems.
Applicable Functions: None
Affects Swat: No
Affects Capacity: No
Affects Geography: No
Additional Remarks:

Instance 'Goal'(25)
Name: Prevent Escalatory Responses
Description: The C2 System shall base decision and control processes on the current rules of engagement allowing sufficient manual control to be maintain and preventing inappropriate responses in situations potentially leading to war escalation.
Goal Group: Amount of Manual Participation
Satisfactory: SATISFACTORY - The C2 System provides analysis of the threat situation and sufficient information for the man to consider the appropriateness of alternative response actions.
Partially Satisfactory: Partial Satisfaction - The C2 System is designed in such a way that an operator, without proper command authority, could under some circumstances select actions that would escalate the state of the war.
Unsatisfactory: UNSATISFACTORY - The C2 System was designed to automatically engage the weapon system without consideration of the political situation.
Not Supported: NOT SUPPORTED - This goal is not supported if there is insufficient communications of political status or threat assessment status to allow informed military/political decision making.
Not Applicable: NOT APPLICABLE - This goal is not applicable in conditions where the system has been designed to always perform unconstrained engagement.
Applicable Functions: None
Affects Swat: No
Affects Capacity: No
Affects Geography: No
Additional Remarks:

Instance 'Goal'(26)
Name: No Single Operator Can Disable System
Description: No single system operator shall be capable of disabling the system or causing the system to take undesired action.
Goal Group: System Security and Fault Tolerance
Satisfactory: SATISFACTORY - The C2 System is designed is such a way that command authority is maintained in the decision process and that individual operator action can be detected and overridden at higher authority levels.
Partially Satisfactory: PARTIALLY SATISFACTORY - The C2 System is designed in such a way that most activities of a particular operator can be detected and overridden.

Unsatisfactory: UNSATISFACTORY - The C2 System is designed in such a way that one operator can, without detection, cause a major loss in system performance by electing inappropriate or no activity.
Not Supported: NOT SUPPORTED - This goal can be supported within the C2 System.
Not Applicable: NOT APPLICABLE - This goal is applicable to all C2 Systems.
Applicable Functions: [System Monitoring/Recovery,Operator Input Processing]
Affects Swat: No
Affects Capacity: No
Affects Geography: No
Additional Remarks: [Secure Data Processing]

Instance 'Goal'(27)
Name: No Program Can Disable
Description: No single system C2 programmer should be capable of establishing a program that can disable a system or cause a system to take undesired actions.
Goal Group: System Security and Fault Tolerance
Satisfactory: SATISFACTORY - Software within the C2 System has been designed and tested to ensure that there are no provisions in the software code that could cause a system to not operate, or to take an inappropriate response.
Partially Satisfactory: PARTIALLY SATISFACTORY - The software system has been designed in a hierarchical top-down manner to allow reasonable confidence that it will perform correctly despite ill-intentions of any system programmer.
Unsatisfactory: UNSATISFACTORY - The C2 System has evolved through complex software changes and there is sufficient review and testing of the software so that there is not a high confidence that the system will perform inappropriately if a malicious software fault has been included.
Not Supported: NOT SUPPORTED - This goal can be supported within the C2 System.
Not Applicable: NOT APPLICABLE - This goal is not applicable if the C2 System does not include software.
Applicable Functions: [System Monitoring/Recovery]
Affects Swat: No
Affects Capacity: No
Affects Geography: No
Additional Remarks: [Secure Data Processing]

Instance 'Goal'(28)
Name: Releases/Roe's Protected
Description: All releases and rules of engagement shall be protected within the C2 System.
Goal Group: System Security and Fault Tolerance
Satisfactory: SATISFACTORY - The C2 System has been designed so that the enemy may not infer the current release status or rules of engagement without incurring substantial penalty.
Partially Satisfactory: PARTIALLY SATISFACTORY - The C2 System has been designed in such a way that they enemy may, by expending his resources, perform an experiment that will determine the release state and rules of engagement being employed so that he may exploit these in the future. However, this information will come at high cost to the enemy.
Unsatisfactory: UNSATISFACTORY - The C2 System has been designed in such a way that the enemy may perform experiments that will provide him with information of the likely release status and rules of engagement that can be exploited in a future engagement.
Not Supported: NOT SUPPORTED - This goal can be supported within the C2 System.
Not Applicable: NOT APPLICABLE - This goal is applicable to all C2 Systems.
Applicable Functions: None

Affects Swat: No
Affects Capacity: No
Affects Geography: No
Additional Remarks: [Physical Security,Survivable Communications,Secure Data Processing]

Instance 'Goal'(29)

Name: Redundancy/Backup

Description: The system shall have redundancy and backup features to assume backup responsibility in the event of an online system element failure.

Goal Group: System Security and Fault Tolerance

Satisfactory: SATISFACTORY - This system has been designed so that no combination of failures that are more likely than .01 will degrade the system operation.

Partially Satisfactory: PARTIALLY SATISFACTORY - This system has been designed with sufficient backup capability so that a minor degradation in the system performance may occur under selected failure modes, but the system mission will not be substantially degraded.

Unsatisfactory: UNSATISFACTORY - The system has been designed in such a way that a likely collection (closer than These faults may be increased in likelihood by threat action.

Not Supported: NOT SUPPORTED - This goal can be supported within the C2 System.

Not Applicable: NOT APPLICABLE - This goal is applicable to all C2 Systems.

Applicable Functions: [Real Time Control,System Monitoring/Recovery]

Affects Swat: No
Affects Capacity: No
Affects Geography: No
Additional Remarks:

s. Class: *Goal Group*

Instance 'Goal Group'(1)
    Name: System Robustness
    Description: System Robustness:
        The System Robustness Goals define the desired system capability with respect to the ability to defend against all types of threats under all probable conditions of attack.

Instance 'Goal Group'(2)
    Name: Surveillance Data Use
    Description: Surveillance Data Use:
        The Surveillance Data Use Goals define the desired system capability with respect to the obtaining the maximum amount of information from the received surveillance data.

Instance 'Goal Group'(3)
    Name: System Response Selection
    Description: System Response Selection:
        The System Response Selection Goals define the desired system capability with respect to the utilizing the most effective and efficient weapon to negate the threat.

Instance 'Goal Group'(4)
    Name: System Timeliness
    Description: System Timeliness:
        The System Timeliness Goals define the desired system capability with respect to the response time associated with decision making leading to defensive action.

Instance 'Goal Group'(5)
    Name: System Authority Control
    Description: System Authority Control:
        The System Authority Control Goals define the desired system capability with respect to maintaining an orderly flow of data and commands at all times.

Instance 'Goal Group'(6)
    Name: System Survivability
    Description: System Survivability:
        The System Survivability Goals define the desired system capability with respect to the system surviving an attack, maintaining continuity of operations, and regrouping to counter subsequent attacks.

Instance 'Goal Group'(7)
    Name: Amount of Manual Participation
    Description: Amount of Manual Participation:
        The Amount of Manual Participation Goals define the desired system capability with respect to maintaining a man-in-the- loop for control while not overwhelming the operator with too much data.

Instance 'Goal Group'(8)
    Name: System Security and Fault Tolerance
    Description: System Security and Fault Tolerance:
        The System Security and Fault Tolerance Goals define the desired system capability with respect to ensuring the security of the operation of the system and the data contained within it.

6. Class: *Rule*

Instance 'Rule'(1)
Excursion Element: Threats
Excursion Characteristic: Number
Excursion Change: Increase
Impact Element: Sensors
Impact Characteristic: Number
Impact Change: Increase
Rationale: Ensure sensors exist through all phases of a battle.
Notes: Since active sensor assets are expected to be attacked, an inactive set of sensors brought on-line as primary sensors are lost may contribute favorably to achieving the necessary coverage.
Applicable Goals: [Maximum Case Attack,Correlate & Fuse Data,Control Resources,Support Engagements,Give Updates to Weapons,Graceful Degradation]

Instance 'Rule'(2)
Excursion Element: Threats
Excursion Characteristic: Number
Excursion Change: Increase
Impact Element: Sensors
Impact Characteristic: Capacity
Impact Change: Increase
Rationale: Ensure individual sensor capacities are not exceeded.
Notes: Since a greater number of threats are expected in the battle, individual sensor capacities may need to be increased or sensors may be required to have reduced areas of coverage to prevent saturation.
Applicable Goals: [Maximum Case Attack,Correlate & Fuse Data,Control Resources,Support Engagements,Give Updates to Weapons,Graceful Degradation]

Instance 'Rule'(3)
Excursion Element: Threats
Excursion Characteristic: Number
Excursion Change: Increase
Impact Element: Sensors
Impact Characteristic: Survivability
Impact Change: Increase
Rationale: Since sensors are expected to be the targets of an attack increasing survivability could ensure sensors are available throughout all phases of the attack, e.g. to provide early warning of follow-on attacks.
Notes: Since sensor assets are expected to be attacked; counter-measures and passive defense may be an effective means of preserving sensors necessary to conduct surveillance and response.
Applicable Goals: [Maximum Case Attack,Correlate & Fuse Data,Control Resources,Support Engagements,Give Updates to Weapons,Graceful Degradation]

Instance 'Rule'(4)
Excursion Element: Threats
Excursion Characteristic: Spatial Density
Excursion Change: Increase
Impact Element: Sensors
Impact Characteristic: Resolution
Impact Change: Increase
Rationale: Rationale: to ensure that individual targets can be detected and accurate raid counts can be made.
Notes: Although higher densities of targets may be easier to detect as a whole, accurate raid counts and positions of individual targets may be necessary for targetting weapon systems.
Applicable Goals: [Use all Data Received,Correlate & Fuse Data,Control Resources]

Instance 'Rule'(5)
Excursion Element: Threats
Excursion Characteristic: Speed
Excursion Change: Increase
Impact Element: Sensors
Impact Characteristic: Location
Impact Change: Different
Rationale: To ensure there is sufficient time for defensive systems to react the sensors may need to be moved closer, if possible, to the threat location.
Notes: In a case where the IF condition of this statement is true, there is assumed to be a shorter reaction time because of the increased speed of the threat thereby making early detection of the enemy imperative.
Applicable Goals: [Control Resources,Multiple Weapons Commitments,Send Data & Directives in Sufficient Time]

Instance 'Rule'(6)
Excursion Element: Threats
Excursion Characteristic: Speed
Excursion Change: Increase
Impact Element: Sensors
Impact Characteristic: Coverage Volume
Impact Change: Increase
Rationale: To ensure there is sufficient time for defensive systems to react, assuming the sensor location is a constant, the effective coverage volume or maximum detection range of the sensor will need to be increased so that threat detection can take place at greater ranges from the protected area.
Notes: In many cases it may not be possible or practical to attempt to increase the detection range and thus the coverage volume of the sensor, the applicability of this concept therefore depends on the type of sensor at its particular characteristics.
Applicable Goals: [Control Resources,Multiple Weapons Commitments,Send Data & Directives in Sufficient Time]

Instance 'Rule'(7)
Excursion Element: Threats
Excursion Characteristic: Speed
Excursion Change: Increase
Impact Element: Sensors
Impact Characteristic: Scan Rate
Impact Change: Increase
Rationale: Threats moving at a higher than expected rate of speed will obviously cover greater distances in a shorter time period thus making it necessary to re-visit their flight path more often to determine their exact location and proximity to the protected area.
Notes: Higher scan rates for existing sensors may not be possible or practical depending on the sensor type. Additionally, higher scan rates generally mean a shorter maximum range which may be detrimental in this case.
Applicable Goals: [Control Resources,Multiple Weapons Commitments,Send Data & Directives in Sufficient Time]

Instance 'Rule'(8)
Excursion Element: Threats
Excursion Characteristic: Detectability
Excursion Change: Decrease
Impact Element: Sensors
Impact Characteristic: Sensitivity
Impact Change: Increase
Rationale: In order to provide detection capability at the ranges comparable to that which is achievable with threats of normal sensor signature or ECM capability, more sensitive sensors are necessary to detect the threat.
Notes: Increasing the sensitivity of fielded systems may not be practical or possible and a new sensor design concept may be necessary.

Applicable Goals: [Use all Data Received,Correlate & Fuse Data,Control Resources,Send Data & Directives in Sufficient Time]

Instance 'Rule'(9)
Excursion Element: Threats
Excursion Characteristic: Detectability
Excursion Change: Decrease
Impact Element: Sensors
Impact Characteristic: Type
Impact Change: Different
Rationale: A sensor with a different signature characteristic may be used to replace or augment the sensor system which now has difficulty detecting the threat due to a decrease in its signature.
Notes: An integrated sensor suite composed of sensors with sensitivities in different frequency bands (such as radar and IR) could be used to provide a much higher data confidence than a single sensor.
Applicable Goals: [Use all Data Received,Correlate & Fuse Data,Control Resources,Send Data & Directives in Sufficient Time]

Instance 'Rule'(10)
Excursion Element: Threats
Excursion Characteristic: Maneuverability
Excursion Change: Increase
Impact Element: Sensors
Impact Characteristic: Sensitivity
Impact Change: Increase
Rationale: A target engaged in maneuvers is continually presenting the sensor with a continuously changing image to detect, thereby frustrating detection logic within the sensor.
Notes: Increasing the sensitivity of fielded systems may not be practical or possible and a new sensor design concept may be necessary. Maneuvering targets represent a 'special class of problem for sensors and as a result C2, continuous data is required on a maneuvering target so that C2 can maintain an accurate position and velocity.
Applicable Goals: [Use all Data Received,Correlate & Fuse Data,Control Resources,Give Updates to Weapons]

Instance 'Rule'(11)
Excursion Element: Threats
Excursion Characteristic: Location (Primary basing)
Excursion Change: Different
Impact Element: Sensors
Impact Characteristic: Location
Impact Change: Different
Rationale: Assuming the Threat moves closer to the protected area, the sensors may need to be moved closer, if possible, to ensure there is sufficient time for defensive systems to react to the threat location.
Notes: There is assumed to be a shorter reaction time because of the closer proximity of the threat thereby making early detection of the enemy imperative. In other cases, where the threat may be relocated to a location where activity is currently out of LOS sensor systems may moved, if possible, to maintain LOS.
Applicable Goals: [Processing for all Regions,Not Rely on Deniable Data,Give Updates to Weapons,Send Data & Directives in Sufficient Time]

Instance 'Rule'(12)
Excursion Element: Threats
Excursion Characteristic: Location (Primary basing)
Excursion Change: Different
Impact Element: Sensors
Impact Characteristic: Type

Impact Change: Different
Rationale: The objective of this effort is to ensure continuous monitoring of threat activity is available such that there is sufficient time for defensive systems to react.
Notes: Assuming the Threat moves to an area currently out of LOS of the existing sensors used to monitor activity new sensor assets may need to be directed into the area, such as spaced based assets or ELINT assets which do not have LOS restrictions.
Applicable Goals: [Processing for all Regions,Not Rely on Deniable Data,Give Updates to Weapons,Send Data & Directives in Sufficient Time]

Instance 'Rule'(13)
Excursion Element: Threats
Excursion Characteristic: Characteristic(34)
Excursion Change: Decrease
Impact Element: Sensors
Impact Characteristic: Resolution
Impact Change: Increase
Rationale: Depending upon the type of sensor used to perform identification of targets, tightening the tolerances and providing an increase in the resolution may yield the ability to detect anomalies in the detected signal permitting positive identification.
Notes: This scheme may not be possible or practical for many sensor type and alternate identification means may need to be utilized.
Applicable Goals: [Use all Data Received,Not Rely on Deniable Data,Correlate & Fuse Data,Control Resources,Send Data & Directives in Sufficient Time]

Instance 'Rule'(14)
Excursion Element: Threats
Excursion Characteristic: Characteristic(34)
Excursion Change: Decrease
Impact Element: Sensors
Impact Characteristic: Type
Impact Change: Different
Rationale: Decreased identifiability on the part of one type of sensor may not represent a change in identifiability on the part of other sensor types operating in different frequency bands.
Notes: This scheme may require the use of sensor assets which may be fully committed to other areas of responsibility, therefore time-sharing of some sensors for other purposes may be required.
Applicable Goals: [Use all Data Received,Not Rely on Deniable Data,Correlate & Fuse Data,Control Resources,Send Data & Directives in Sufficient Time]

Instance 'Rule'(15)
Excursion Element: Threats
Excursion Characteristic: Number
Excursion Change: Increase
Impact Element: Weapons
Impact Characteristic: Number
Impact Change: Increase
Rationale: Ensure that the weapon system are not overwhelmed by a numerically superior force.
Notes: In some cases, the weapon systems may be superior even though they are smaller in numbers. Adding more weapons to the battle could increase the loads on the C2 facility.
Applicable Goals: [Maximum Case Attack,Conserve Defensive Resources,Expend Resources Proportionally]

Instance 'Rule'(16)
Excursion Element: Threats
Excursion Characteristic: Spatial Density

Excursion Change: Increase
Impact Element: Weapons
Impact Characteristic: Effective Volume
Impact Change: Increase
Rationale: Provide the capability to intercept and destroy the threat at longer ranges so that the higher density of targets is thinned out greater distance from the protected area ensuring greater survivability of individual assets.
Notes: Eliminating threats at greater ranges reduces the possibility of C2 system saturation and weapon conflicts that would occur due in a close in battle. Existing weapon systems may have lower Pk's associated with longer range intercepts and new weapon systems or modifications to existing weapon systems may be necessary.
Applicable Goals: [Expend Resources Proportionally,Optimize Time to Intercept,Environmental & Geometric Conditions]

Instance 'Rule'(17)
Excursion Element: Threats
Excursion Characteristic: Spatial Density
Excursion Change: Increase
Impact Element: Weapons
Impact Characteristic: Probability of Kill
Impact Change: Different
Rationale: Increasing the Pk translates to fewer shots required to eliminate an equivalent number of targets, therefore providing the time needed to engage more of the threats before they can threaten assets.
Notes: In target rich environments, assuming all targets are of equal value, fire and forget weapons or those with proximity fuses and large areas of effectiveness could potentially provide a more efficient defense.
Applicable Goals: [Expend Resources Proportionally,Optimize Time to Intercept,Environmental & Geometric Conditions]

Instance 'Rule'(18)
Excursion Element: Threats
Excursion Characteristic: Spatial Density
Excursion Change: Increase
Impact Element: Weapons
Impact Characteristic: Firing Rate
Impact Change: Increase
Rationale: To ensure the weapon systems have the ability to deal with large numbers of threat over a short period of time without exhausting their total shots and requiring down time to reload.
Notes: Increasing the Firing Rate/Number of Shots may not be possible or practical for many existing weapon systems.
Applicable Goals: [Expend Resources Proportionally,Optimize Time to Intercept,Environmental & Geometric Conditions]

Instance 'Rule'(19)
Excursion Element: Threats
Excursion Characteristic: Spatial Density
Excursion Change: Increase
Impact Element: Weapons
Impact Characteristic: Type
Impact Change: Different
Rationale: Weapons currently being may be saturated with the higher spatial density of the threat; utilizing different weapon types (DEWs vs. Aircraft and SAMs or SAMs vs. Aircraft) may have the ability to deal with the threat easier
Notes: Changing the type of weapon may not be feasible because of basing requirements or technology that is fully developed.
Applicable Goals: [Expend Resources Proportionally,Optimize Time to Intercept,Environmental & Geometric Conditions]

Instance 'Rule'(20)

Excursion Element: Threats
Excursion Characteristic: Speed
Excursion Change: Increase
Impact Element: Weapons
Impact Characteristic: Speed
Impact Change: Increase
Rationale: Conventional intercepts with a speed disadvantage require precise timing and geometry to execute and increases the rate of failure of the intercept due to the ability of the threat to "outrun" the interceptor.
Notes: A classic example of this condition is exemplified by the many attempts of the Soviets to down SR-71 reconnaissance aircraft without a weapon system capable of equivalent speeds.
Applicable Goals: [Consider Assets and Risk,Optimize Time to Intercept,Environmental & Geometric Conditions,Send Data & Directives in Sufficient Time]

Instance 'Rule'(21)
Excursion Element: Threats
Excursion Characteristic: Speed
Excursion Change: Increase
Impact Element: Weapons
Impact Characteristic: Type
Impact Change: Different
Rationale: Weapons currently being used to defend against a threat may no longer be effective, since an intercept may be outside or on the fringes of their performance envelope, therefore new weapons systems such as Directed Energy Weapons (DEWs) may need to be employed.
Notes: New weapon systems may need to be developed to defend against threat that are outside performance envelopes of current weapons.
Applicable Goals: [Consider Assets and Risk,Optimize Time to Intercept,Environmental & Geometric Conditions,Send Data & Directives in Sufficient Time]

Instance 'Rule'(22)
Excursion Element: Threats
Excursion Characteristic: Maneuverability
Excursion Change: Increase
Impact Element: Weapons
Impact Characteristic: Type
Impact Change: Different
Rationale: Threats with greater maneuverability may have the capability to out-perform the weapons currently utilized to destroy them, therefore employment of specific weapon types and tactics may need to be reconsidered.
Notes: This may require redesign of weapon carriage systems for some aircraft types (if air launched) or may involve re-thinking of tactics (utilizing SAMs as a front-line defense rather than interceptors).
Applicable Goals: [Optimize Time to Intercept,Multiple Weapons Commitments,Environmental & Geometric Conditions,Give Updates to Weapons]

Instance 'Rule'(23)
Excursion Element: Threats
Excursion Characteristic: Detectability
Excursion Change: Decrease
Impact Element: Weapons
Impact Characteristic: Sensitivity
Impact Change: Increase
Rationale: The weapon must be able to detect the threat and follow it to an intercept point with its terminal sensor, therefore the sensitivity should be increased to compensate for the change in threat signature.
Notes: Increasing the sensitivity of fielded systems may not be practical or possible and a new weapon design concept may be

necessary.

Applicable Goals: [Optimize Time to Intercept,Multiple Weapons Commitments,Environmental & Geometric Conditions,Give Updates to Weapons]

Instance 'Rule'(24)

Excursion Element: Threats
Excursion Characteristic: Detectability
Excursion Change: Decrease
Impact Element: Weapons
Impact Characteristic: Type
Impact Change: Different
Rationale: Rationale: a weapon with a different signature characteristic may be used to replace the weapon system which now has difficulty intercepting the threat due to a decrease in its signature.
Notes: An integrated weapon suite composed of multiple sensors with sensitivities in different frequency bands (such as radar and IR) could be used to provide a much higher kill probability than a single sensor in a weapon.
Applicable Goals: [Optimize Time to Intercept,Multiple Weapons Commitments,Environmental & Geometric Conditions,Give Updates to Weapons]

Instance 'Rule'(25)

Excursion Element: Threats
Excursion Characteristic: Identifiability
Excursion Change: Increase
Impact Element: Weapons
Impact Characteristic: Number
Impact Change: Increase
Rationale: An increase in the number of shots will most likely be required to eliminate the threat since the overall Pk will be lower as a result of the increased threat capability.
Notes: Increasing the weapons in battle may have a tendency to increase the overall load on the C2.
Applicable Goals: [Multiple Weapons Commitments,Environmental & Geometric Conditions,Send Data & Directives in Sufficient Time]

Instance 'Rule'(26)

Excursion Element: Threats
Excursion Characteristic: Identifiability
Excursion Change: Increase
Impact Element: Weapons
Impact Characteristic: Probability of Kill
Impact Change: Different
Rationale: The enhanced weapon avoidance capability of the threat will require more sophisticated or different techniques on the part of the weapon terminal guidance sensor and system to prevent the threat from escaping.
Notes: The weapons and tactics utilized to combat a particular threat may need revision in light of enhanced capabilities.
Applicable Goals: [Multiple Weapons Commitments,Environmental & Geometric Conditions,Send Data & Directives in Sufficient Time]

Instance 'Rule'(27)

Excursion Element: Threats
Excursion Characteristic: Identifiability
Excursion Change: Increase
Impact Element: Weapons
Impact Characteristic: Type
Impact Change: Different
Rationale: A revision of the weapons that are designated for use against a particular threat type may need to occur due to enhanced capabilities and the unique features of each weapon type.
Notes: A weapon type with a sensor that is not impacted by

the threat enhancements should be chosen to be used in defense against the threat.
Applicable Goals: [Multiple Weapons Commitments,Environmental & Geometric Conditions,Send Data & Directives in Sufficient Time]

Instance 'Rule'(28)

Excursion Element: Threats
Excursion Characteristic: Identifiability
Excursion Change: Increase
Impact Element: Weapons
Impact Characteristic: Firing Rate
Impact Change: Increase
Rationale: An increase in the number of shots will most likely be required to eliminate the threat since the overall Pk will be lower as a result of the increased threat capability.
Notes: Increasing the Firing Rate/Number of Shots may not be possible or practical for many existing weapon systems.
Applicable Goals: [Multiple Weapons Commitments,Environmental & Geometric Conditions,Send Data & Directives in Sufficient Time]

Instance 'Rule'(29)

Excursion Element: Threats
Excursion Characteristic: Location (Primary basing)
Excursion Change: Different
Impact Element: Weapons
Impact Characteristic: Location
Impact Change: Different
Rationale: Assuming the threat has moved closer, to provide an equivalent reaction time, such that intercepts can be performed at equivalent distances from the protected area
Notes: Movement of large numbers, stockpiles of weapons and their supporting equipment and forces may not be practical due to geography or location availability.
Applicable Goals: [Optimize Time to Intercept,Multiple Weapons Commitments,Environmental & Geometric Conditions]

Instance 'Rule'(30)

Excursion Element: Sensors
Excursion Characteristic: Number
Excursion Change: Increase
Impact Element: C2
Impact Characteristic: Sensor Control
Impact Change: Different
Rationale: Increasing the number of sensors will lead to more data to process by these functions.
Notes: None
Applicable Goals: None

Instance 'Rule'(31)

Excursion Element: Sensors
Excursion Characteristic: Number
Excursion Change: Increase
Impact Element: C2
Impact Characteristic: Tracking
Impact Change: Different
Rationale: Increasing the number of sensors will lead to more data to process by these functions.
Notes: None
Applicable Goals: None

Instance 'Rule'(32)

Excursion Element: Sensors
Excursion Characteristic: Location
Excursion Change: Different
Impact Element: C2
Impact Characteristic: Sensor Control

Impact Change: Different
Rationale: Sensor location changes will impact the processing performed for blanking control, radiation management, and ECCM control because of coverage changes and the registration process due to location changes.
Notes: None
Applicable Goals: None

Instance 'Rule'(33)
Excursion Element: Sensors
Excursion Characteristic: Location
Excursion Change: Different
Impact Element: C2
Impact Characteristic: Tracking
Impact Change: Different
Rationale: Sensor location changes will impact the processing performed for blanking control, radiation management, and ECCM control because of coverage changes and the registration process due to location changes.
Notes: None
Applicable Goals: None

Instance 'Rule'(34)
Excursion Element: Sensors
Excursion Characteristic: Type
Excursion Change: Different
Impact Element: C2
Impact Characteristic: Sensor Control
Impact Change: Different
Rationale: The position information, and error distributions of the various sensor types are likely to be very different from each other requiring unique schemes for processing the data.
Notes: None
Applicable Goals: None

Instance 'Rule'(35)
Excursion Element: Sensors
Excursion Characteristic: Type
Excursion Change: Different
Impact Element: C2
Impact Characteristic: Tracking
Impact Change: Different
Rationale: The position information, and error distributions of the various sensor types are likely to be very different from each other requiring unique schemes for processing the data.
Notes: None
Applicable Goals: None

Instance 'Rule'(36)
Excursion Element: Sensors
Excursion Characteristic: Type
Excursion Change: Different
Impact Element: C2
Impact Characteristic: Threat Evaluation
Impact Change: Different
Rationale: The position information, and error distributions of the various sensor types are likely to be very different from each other requiring unique schemes for processing the data.
Notes: None
Applicable Goals: None

Instance 'Rule'(37)
Excursion Element: Sensors
Excursion Characteristic: Type
Excursion Change: Different
Impact Element: C2
Impact Characteristic: Threat Assessment/Identification
Impact Change: Different
Rationale: The position information, and error distributions of

the various sensor types are likely to be very different from each other requiring unique schemes for processing the data.
Notes: None
Applicable Goals: None

Instance 'Rule'(38)
Excursion Element: Sensors
Excursion Characteristic: Coverage Volume
Excursion Change: Different
Impact Element: C2
Impact Characteristic: Sensor Control
Impact Change: Different
Rationale: Due to the increased coverage volume more target reports are anticipated and will have to be processed.
Notes: None
Applicable Goals: None

Instance 'Rule'(98)
Excursion Element: Sensors
Excursion Characteristic: Coverage Volume
Excursion Change: Different
Impact Element: C2
Impact Characteristic: Tracking
Impact Change: Different
Rationale: Due to the increased coverage volume more target reports are anticipated and will have to be processed.
Notes: None
Applicable Goals: None

Instance 'Rule'(40)
Excursion Element: Sensors
Excursion Characteristic: Sensitivity
Excursion Change: Increase
Impact Element: C2
Impact Characteristic: Sensor Control
Impact Change: Different
Rationale: with the increased sensitivity it is assumed that more target reports will be received by the C2 system therefore increasing the processing load accordingly.
Notes: None
Applicable Goals: None

Instance 'Rule'(41)
Excursion Element: Sensors
Excursion Characteristic: Sensitivity
Excursion Change: Increase
Impact Element: C2
Impact Characteristic: Tracking
Impact Change: Different
Rationale: with the increased sensitivity it is assumed that more target reports will be received by the C2 system therefore increasing the processing load accordingly.
Notes: None
Applicable Goals: None

Instance 'Rule'(42)
Excursion Element: Sensors
Excursion Characteristic: Resolution
Excursion Change: Increase
Impact Element: C2
Impact Characteristic: Sensor Control
Impact Change: Different
Rationale: The increase in resolution is assumed to result in a greater number of targets reported to the C2 system therefore increasing the processing load accordingly.
Notes: None
Applicable Goals: None

Instance 'Rule'(43)

Excursion Element: Sensors
Excursion Characteristic: Resolution
Excursion Change: Increase
Impact Element: C2
Impact Characteristic: Tracking
Impact Change: Different
Rationale: The increase in resolution is assumed to result in a greater number of targets reported to the C2 system therefore increasing the processing load accordingly.
Notes: None
Applicable Goals: None

Instance 'Rule'(44)
Excursion Element: Sensors
Excursion Characteristic: Survivability
Excursion Change: Increase
Impact Element: C2
Impact Characteristic: Sensor Control
Impact Change: Different
Rationale: Assuming the method used to increase survivability is through radiation management then the C2 system will be required to perform more processing to provide protection for the sensor by continuously altering sensor coverage areas.
Notes: None
Applicable Goals: None

Instance 'Rule'(45)
Excursion Element: Sensors
Excursion Characteristic: Capacity
Excursion Change: Different
Impact Element: C2
Impact Characteristic: Tracking
Impact Change: Different
Rationale: Assuming an increase in capacity, more target will be reported and more processing will be required on the part of the C2 system otherwise, no increase in processing will occur.
Notes: None
Applicable Goals: None

Instance 'Rule'(46)
Excursion Element: Sensors
Excursion Characteristic: Scan Rate
Excursion Change: Different
Impact Element: C2
Impact Characteristic: Tracking
Impact Change: Different
Rationale: Changes in the sensor scan rate translate into not only the quantity of data received by the C2 system but also the timing associated with it, increasing the scan rate, for example, may require the C2 system to update its database more frequently.
Notes: None
Applicable Goals: None

Instance 'Rule'(47)
Excursion Element: Weapons
Excursion Characteristic: Number
Excursion Change: Increase
Impact Element: C2
Impact Characteristic: Tracking
Impact Change: Different
Rationale: The C2 system will have more objects to track, identify and report to other C2 nodes as well as have a greater number of weapons from which to select from and control for intercepts.
Notes: None
Applicable Goals: None

Instance 'Rule'(48)

Excursion Element: Weapons
Excursion Characteristic: Number
Excursion Change: Increase
Impact Element: C2
Impact Characteristic: Threat Assessment/Identification
Impact Change: Different
Rationale: The C2 system will have more objects to track, identify and report to other C2 nodes as well as have a greater number of weapons from which to select from and control for intercepts.
Notes: None
Applicable Goals: None

Instance 'Rule'(49)
Excursion Element: Weapons
Excursion Characteristic: Number
Excursion Change: Increase
Impact Element: C2
Impact Characteristic: Weapon Assignment
Impact Change: Different
Rationale: The C2 system will have more objects to track, identify and report to other C2 nodes as well as have a greater number of weapons from which to select from and control for intercepts.
Notes: None
Applicable Goals: None

Instance 'Rule'(50)
Excursion Element: Weapons
Excursion Characteristic: Number
Excursion Change: Increase
Impact Element: C2
Impact Characteristic: Weapon Control
Impact Change: Different
Rationale: The C2 system will have more objects to track, identify and report to other C2 nodes as well as have a greater number of weapons from which to select from and control for intercepts.
Notes: None
Applicable Goals: None

Instance 'Rule'(51)
Excursion Element: Weapons
Excursion Characteristic: Number
Excursion Change: Increase
Impact Element: C2
Impact Characteristic: Telling
Impact Change: Different
Rationale: The C2 system will have more objects to track, identify and report to other C2 nodes as well as have a greater number of weapons from which to select from and control for intercepts.
Notes: None
Applicable Goals: None

Instance 'Rule'(52)
Excursion Element: Weapons
Excursion Characteristic: Type
Excursion Change: Different
Impact Element: C2
Impact Characteristic: Weapon Assignment
Impact Change: Different
Rationale: The C2 system will have to have provisions made for assignment and control of the new weapon types and their roles. The provisions may be in the form of decision logic or algorithm changes which can impact the system processing load.
Notes: None
Applicable Goals: None

Instance 'Rule'(53)
  Excursion Element: Weapons
  Excursion Characteristic: Type
  Excursion Change: Different
  Impact Element: C2
  Impact Characteristic: Weapon Control
  Impact Change: Different
  Rationale: The C2 system will have to have provisions made for assignment and control of the new weapon types and their roles. The provisions may be in the form of decision logic or algorithm changes which can impact the system processing load.
  Notes: None
  Applicable Goals: None

Instance 'Rule'(54)
  Excursion Element: Weapons
  Excursion Characteristic: Effective Volume
  Excursion Change: Different
  Impact Element: C2
  Impact Characteristic: Sensor Control
  Impact Change: Different
  Rationale: The C2 system will be required to maintain surveillance on the long range intercepts as well as compute and control longer and potentially more complex intercept paths.
  Notes: None
  Applicable Goals: None

Instance 'Rule'(55)
  Excursion Element: Weapons
  Excursion Characteristic: Effective Volume
  Excursion Change: Different
  Impact Element: C2
  Impact Characteristic: Weapon Assignment
  Impact Change: Different
  Rationale: The C2 system will be required to maintain surveillance on the long range intercepts as well as compute and control longer and potentially more complex intercept paths.
  Notes: None
  Applicable Goals: None

Instance 'Rule'(56)
  Excursion Element: Weapons
  Excursion Characteristic: Effective Volume
  Excursion Change: Different
  Impact Element: C2
  Impact Characteristic: Weapon Control
  Impact Change: Different
  Rationale: The C2 system will be required to maintain surveillance on the long range intercepts as well as compute and control longer and potentially more complex intercept paths.
  Notes: None
  Applicable Goals: None

Instance 'Rule'(57)
  Excursion Element: Weapons
  Excursion Characteristic: Sensitivity
  Excursion Change: Different
  Impact Element: C2
  Impact Characteristic: Weapon Assignment
  Impact Change: Different
  Rationale: The C2 system will have to consider the resultant effect of the sensitivity change to the weapon in terms of the selection of weapon for an intercept, tactics for approach of the target and proximity of the weapon to target for a kill.
  Notes: None
  Applicable Goals: None

Instance 'Rule'(58)

Excursion Element: Weapons
  Excursion Characteristic: Sensitivity
  Excursion Change: Different
  Impact Element: C2
  Impact Characteristic: Weapon Control
  Impact Change: Different
  Rationale: The C2 system will have to consider the resultant effect of the sensitivity change to the weapon in terms of the selection of weapon for an intercept, tactics for approach of the target and proximity of the weapon to target for a kill.
  Notes: None
  Applicable Goals: None

Instance 'Rule'(59)
  Excursion Element: Weapons
  Excursion Characteristic: Probability of Kill
  Excursion Change: Different
  Impact Element: C2
  Impact Characteristic: Weapon Assignment
  Impact Change: Different
  Rationale: The C2 system will have to consider the resultant effect of the probability of kill change to the weapon in terms of the selection of weapon for an intercept.
  Notes: None
  Applicable Goals: None

Instance 'Rule'(60)
  Excursion Element: Weapons
  Excursion Characteristic: Speed
  Excursion Change: Different
  Impact Element: C2
  Impact Characteristic: Weapon Assignment
  Impact Change: Different
  Rationale: The C2 system will have to consider the resultant effect of the performance capability change to the weapon in terms of the selection of weapon for an intercept and tactics used for the intercept.
  Notes: None
  Applicable Goals: None

Instance 'Rule'(61)
  Excursion Element: Weapons
  Excursion Characteristic: Speed
  Excursion Change: Different
  Impact Element: C2
  Impact Characteristic: Weapon Control
  Impact Change: Different
  Rationale: The C2 system will have to consider the resultant effect of the performance capability change to the weapon in terms of the selection of weapon for an intercept and tactics used for the intercept.
  Notes: None
  Applicable Goals: None

Instance 'Rule'(62)
  Excursion Element: Weapons
  Excursion Characteristic: Firing Rate
  Excursion Change: Different
  Impact Element: C2
  Impact Characteristic: Weapon Assignment
  Impact Change: Different
  Rationale: The C2 system may require upgrades to take advantage of the capability to fire more often and for a greater duration.
  Notes: None
  Applicable Goals: None

Instance 'Rule'(63)
  Excursion Element: Weapons

Excursion Characteristic: Firing Rate
Excursion Change: Different
Impact Element: C2
Impact Characteristic: Weapon Control
Impact Change: Different
Rationale: The C2 system may require upgrades to take advantage of the capability to fire more often and for a greater duration.
Notes: None
Applicable Goals: None

Instance 'Rule'(64)
Excursion Element: Threats
Excursion Characteristic: Number
Excursion Change: Increase
Impact Element: C2
Impact Characteristic: Sensor Control
Impact Change: Different
Rationale: The C2 system will have a larger overall number of items to process, track, identify, assign to weapons and forward to other C2 nodes.
Notes: None
Applicable Goals: None

Instance 'Rule'(65)
Excursion Element: Threats
Excursion Characteristic: Number
Excursion Change: Increase
Impact Element: C2
Impact Characteristic: Tracking
Impact Change: Different
Rationale: The C2 system will have a larger overall number of items to process, track, identify, assign to weapons and forward to other C2 nodes.
Notes: None
Applicable Goals: None

- Instance 'Rule'(66)
Excursion Element: Threats
Excursion Characteristic: Number
Excursion Change: Increase
Impact Element: C2
Impact Characteristic: Threat Assessment/Identification
Impact Change: Different
Rationale: The C2 system will have a larger overall number of items to process, track, identify, assign to weapons and forward to other C2 nodes.
Notes: None
Applicable Goals: None

Instance 'Rule'(67)
Excursion Element: Threats
Excursion Characteristic: Number
Excursion Change: Increase
Impact Element: C2
Impact Characteristic: Weapon Assignment
Impact Change: Different
Rationale: The C2 system will have a larger overall number of items to process, track, identify, assign to weapons and forward to other C2 nodes.
Notes: None
Applicable Goals: None

Instance 'Rule'(68)
Excursion Element: Threats
Excursion Characteristic: Number
Excursion Change: Increase
Impact Element: C2
Impact Characteristic: Telling

Impact Change: Different
Rationale: The C2 system will have a larger overall number of items to process, track, identify, assign to weapons and forward to other C2 nodes.
Notes: None
Applicable Goals: None

Instance 'Rule'(69)
Excursion Element: Threats
Excursion Characteristic: Spatial Density
Excursion Change: Increase
Impact Element: C2
Impact Characteristic: Sensor Control
Impact Change: Different
Rationale: It will be necessary for the C2 system to command more sensor resources to observe the threat to determine its composition and then utilize the additional sensor data to track the objects.
Notes: None
Applicable Goals: None

Instance 'Rule'(70)
Excursion Element: Threats
Excursion Characteristic: Spatial Density
Excursion Change: Increase
Impact Element: C2
Impact Characteristic: Tracking
Impact Change: Different
Rationale: It will be necessary for the C2 system to command more sensor resources to observe the threat to determine its composition and then utilize the additional sensor data to track the objects.
Notes: None
Applicable Goals: None

Instance 'Rule'(71)
Excursion Element: Threats
Excursion Characteristic: Speed
Excursion Change: Increase
Impact Element: C2
Impact Characteristic: Tracking
Impact Change: Different
Rationale: The C2 functions will be required to respond more quickly to the threat to intercept and neutralize it before it can penetrate the protected areas defenses.
Notes: None
Applicable Goals: None

Instance 'Rule'(72)
Excursion Element: Threats
Excursion Characteristic: Speed
Excursion Change: Increase
Impact Element: C2
Impact Characteristic: Weapon Assignment
Impact Change: Different
Rationale: The C2 functions will be required to respond more quickly to the threat to intercept and neutralize it before it can penetrate the protected areas defenses.
Notes: None
Applicable Goals: None

Instance 'Rule'(73)
Excursion Element: Threats
Excursion Characteristic: Speed
Excursion Change: Increase
Impact Element: C2
Impact Characteristic: Weapon Control
Impact Change: Different
Rationale: The C2 functions will be required to respond more

quickly to the threat to intercept and neutralize it before it can penetrate the protected areas defenses.
Notes: None
Applicable Goals: None

Instance 'Rule'(74)
Excursion Element: Threats
Excursion Characteristic: Detectability
Excursion Change: Decrease
Impact Element: C2
Impact Characteristic: Sensor Control
Impact Change: Different
Rationale: The C2 system will potentially be required to perform more processing to bring more sensor assets to bear in an effort to search for, detect, and track threats.
Notes: None
Applicable Goals: None

Instance 'Rule'(75)
Excursion Element: Threats
Excursion Characteristic: Detectability
Excursion Change: Decrease
Impact Element: C2
Impact Characteristic: Tracking
Impact Change: Different
Rationale: The C2 system will potentially be required to perform more processing to bring more sensor assets to bear in an effort to search for, detect, and track threats.
Notes: None
Applicable Goals: None

Instance 'Rule'(76)
. Excursion Element: Threats
Excursion Characteristic: Maneuverability
Excursion Change: Increase
Impact Element: C2
Impact Characteristic: Tracking
Impact Change: Different
Rationale: The C2 system will have to perform more processing to maintain a track of a maneuvering target and continually feed information on the latest position to the weapon to complete a successful intercept.
Notes: None
Applicable Goals: None

Instance 'Rule'(77)
Excursion Element: Threats
Excursion Characteristic: Maneuverability
Excursion Change: Increase
Impact Element: C2
Impact Characteristic: Weapon Control
Impact Change: Different
Rationale: The C2 system will have to perform more processing to maintain a track of a maneuvering target and continually feed information on the latest position to the weapon to complete a successful intercept.
Notes: None
Applicable Goals: None

Instance 'Rule'(78)
Excursion Element: Threats
Excursion Characteristic: Weapon Avoidance
Excursion Change: Increase
Impact Element: C2
Impact Characteristic: Weapon Control
Impact Change: Different
Rationale: The single shot probability of kill will be lower resulting the C2 system having to perform shoot-look-shoot processing and having to assign and command more weapons.

Notes: None
Applicable Goals: None

Instance 'Rule'(79)
Excursion Element: Threats
Excursion Characteristic: Weapon Avoidance
Excursion Change: Increase
Impact Element: C2
Impact Characteristic: Weapon Assignment
Impact Change: Different
Rationale: The single shot probability of kill will be lower resulting the C2 system having to perform shoot-look-shoot processing and having to assign and command more weapons.
Notes: None
Applicable Goals: None

Instance 'Rule'(80)
Excursion Element: Threats
Excursion Characteristic: Weapon Avoidance
Excursion Change: Increase
Impact Element: C2
Impact Characteristic: Telling
Impact Change: Different
Rationale: The single shot probability of kill will be lower resulting the C2 system having to perform shoot-look-shoot processing and having to assign and command more weapons.
Notes: None
Applicable Goals: None

Instance 'Rule'(81)
Excursion Element: Threats
Excursion Characteristic: Location (Primary basing)
Excursion Change: Different
Impact Element: C2
Impact Characteristic: Sensor Control
Impact Change: Different
Rationale: The C2 system may be required to reassign sensors to new areas of responsibility or to task specialized sensor groups individually to monitor and maintain accurate surveillance data on the threat
Notes: None
Applicable Goals: None

Instance 'Rule'(82)
Excursion Element: Threats
Excursion Characteristic: Identifiability
Excursion Change: Decrease
Impact Element: C2
Impact Characteristic: Sensor Control
Impact Change: Different
Rationale: The C2 system may be required to task more sensors to view the object and therefore provide more data which must be processed by the Threat Assessment/Identification functions.
Notes: None
Applicable Goals: None

Instance 'Rule'(83)
Excursion Element: Threats
Excursion Characteristic: Identifiability
Excursion Change: Decrease
Impact Element: C2
Impact Characteristic: Threat Assessment/Identification
Impact Change: Different
Rationale: The C2 system may be required to task more sensors to view the object and therefore provide more data which must be processed by the Threat Assessment/Identification functions.
Notes: None

APPENDIX B
CMLP USER'S MANUAL

B1. INTRODUCTION

The CMLP system is intended to aid the designers of C^2 systems during the early stages of the C^2 system life cycle: concept exploration, concept demonstration, and concept validation. It allows a designer to examine existing systems, describe changes in the functions, characteristics, and relationships of their components, and assess the results qualitatively and (to a restricted degree) quantitatively.

The CMLP system has been designed and programmed with an initial set of functions, characteristics, and relationships that can be generalized to many types of defensive systems. The CMLP system also has the capability of accepting user inputs to create and define new functions, characteristics, relationships, and evaluation criteria. This provides the user with the ability to define nearly any type of defensive system and to manipulate the elements in the manner necessary to evaluate the system's performance with respect to *element sensitivities* as well as to user-defined *evaluation criteria*.

B2. THE CMLP MODEL OF C^2 SYSTEMS

To interact with the CMLP system, the user should have an understanding of the model that underlies CMLP. In the paragraphs that follow, we provide a description of the view taken in CMLP of C^2 systems.

The CMLP system is designed to aid the user in the C^2 design process and to qualitatively assess, with the input of the user, the impact on an overall defense system of changes in C^2 system design, supporting element design (sensors and weapons) and changes in the threat. The approach taken in the CMLP system is based on the desire to *complement* traditional simulation representations of C^2 systems.

The overall goal of the CMLP system is to have the flexibility to describe and represent any of the unique C^2 defensive system types. To do this, it is necessary to define the bounds of a C^2 system. Descriptive data for C^2 systems in CMLP are separated into the following component parts: C^2 elements, sensors, weapons, and threats. C^2 elements are defined to be the data analysis and decision-making portion of a defensive (or offensive) system. In other words, C^2 is a nerve center without the capability to observe or interact with its environment. Based on this definition the C^2 system is worthless by itself. The sensor, weapon, and threat components are the environment in which the C^2 element operates. As the names suggest, the threat is the force the C^2 system is designed to protect against, the sensors are responsible for detecting and reporting the threat to the C^2 element, and the weapons are responsible for eliminating the threat based on the direction of the C^2 element. The C^2 element requires communication channels to its sensors and weapons to effectively dispatch and control its assets; for the purposes of CMLP, the communication means are assumed to be inherent between these elements to simplify the model. One element may function as both a sensor and a weapon, e.g., an aircraft may use its on-board instruments to detect a threat (sensor function) and may also attack the threat (weapon function). The same on-board instruments may support sensor functions and weapon delivery.

Each of the defined elements (C^2, sensors, weapons, and threat) have been specified in a computer display *form* that has the inherent flexibility to represent the characteristics or attributes of any instance of the element. The initial set of characteristics of sensors, weapons, and threat supplied with the system are:

Sensor Characteristics
- Number
- Location
- Type (Band)
- Coverage Volume
- Sensitivity
- Resolution
- Survivability
- Capacity
- Scan Rate

Weapon Characteristics
- Number
- Location
- Type (Fusing, Warhead)
- Effective Volume/Range
- Sensitivity
- Probability of Kill
- Maximum Speed/Acceleration
- Firing Rate/Number of Shots

Threat Characteristics
- Number
- Spatial Density
- Arrival Rate
- Maximum Acceleration/Speed
- Detectability
- Maneuverability
- Weapon Avoidance Capability
- Location
- Identifiability

Although these characteristics can, in many cases, be expressed in numerical terms that specifically define the element's capability, numerics alone do not provide sufficient flexibility. Therefore, an individual element's

capability is also defined in terms of the interactions *between elements* through their characteristics. *Subsystem sensitivity rules* describe the capability and behavior of elements as cause and effect relationships. The general form of a rule is *IF "element characteristic" changes (i.e., increases, decreases, or is different) THEN "element characteristic" changes.* The rules also contain fields which describe examples of the interaction and provide usage notes. Through the use of rules in this form, changes impacting defensive (or offensive) system performance can begin to be analyzed in a qualitative sense.

Similarly, C^2 elements are specified in terms of functions and characteristics. The functions for the C^2 elements have been subdivided into groups. The initial set of functions and characteristics is as follows:

Sensor Control Functions
- Blanking Control
- Radiation Management
- ECCM Control
- Sensor Tasking

Tracking Functions
- Sensor Data Acceptance
- Coordinate Transformation
- Registration
- Fusion/Correlation
- Track Update
- Ambiguity Resolution
- Track Initiation
- Kill Assessment

Threat Evaluation Functions
- Raid Composition
- Flight Characteristics
- Strength
- Priority Ranking
- Enemy Order of Battle Maintenance

Identification/Threat Assessment Functions
- Flight Route Correlation
- Route Deviation Alert
- IFF/SIF Processing
- Geographic Determination

- Challenge Processing
- Discrimination

Weapon Assignment Functions
- Weapon Status
- Weapon Probability of Kill
- Weapon Intercept Time
- Weapon Target Assignment

Weapon Control Functions
- Weapon Solution Generation
- Weapon Guidance
- Target Update Processing
- Weapon Effectiveness

Telling Functions
- Data Receipt
- Position Translation
- Data Transmission
- Input/output Filtering
- Reporting Responsibility
- Link Status Reporting
- Alerts/Warning Information
- Authority Control Arbitration

Executive Functions
- Real Time Control
- System Monitoring/Recovery
- Recording
- Simulation
- Operator Input Processing
- Display Generation

C^2 Characteristics
- Degraded Operations
- Mobile
- NBC Hardened
- Physical Security
- Survivability Communications
- Back-up Capability
- Secure Communications
- Secure Data Processing

As in the case of the sensors, weapons, and threats, rules defining C^2 element behavior based upon changes in the other elements are also defined. The initial set of rules developed from these element characteristics forms the foundation of the CMLP system.

Rules describing behavior are also, by themselves, insufficient; some level of overall capability assessment of the defensive system is required. Traditional simulation tech-

niques are generally used to determine a system's capability based on a given attack scenario. In order to provide a low-cost alternative that has utility in the earliest phases of the C^2 system life cycle, a series of idealized goals has been generated that apply to any defensive system. These goals describe in qualitative terms the expectations of a complete system. The goals are subdivided into groups as follows:

Robustness
- Maximum case attack
- Processing for all regions
- Conservation of defensive resources
- Expend resources based on value of assets under attack

Surveillance Data Usage
- Use all received sensor data
- Do not rely on data sources that are deniable by the enemy
- Obtain best estimate of raid strength
- Correlate and fuse data
- Use sensors to monitor missions/resources
- Use sensor to verify engagement results

Weapon Response Selection
- Consider value of assets at risk if defensive action succeeds (i.e., salvage fusing)
- Optimize time to intercept
- Allow multiple weapon commitments
- Consider environmental and geometric conditions
- Use intelligence and doctrinal employment information
- Provide weapons with threat updates

Timeliness
- Send data and directives in time to ensure that defensive posture is established
- Provide for timely recovery from failure
- Provide capability to process maximum data from all sources

Authority Control
- Defined chain of authority
- Maintain data on system performance
- Base decisions on ROE

Survivability
- Operate in all anticipated environments
- Graceful degradation

Degree of Manual Participation
- Balance automated and manual processes
- Allow sufficient manual control to prevent escalation

Security and Fault Tolerance
- No single operator can disable system
- No program can disable system
- All releases and ROEs protected from compromise
- Built in redundancy and backup

Each goal has five defined levels of performance: Satisfactory, Partially Satisfactory, Unsatisfactory, Not Applicable, and Not Supported. These performance levels are intended to provide guidelines to the user when evaluating a particular system.

To make the goals, as a subjective evaluation tool, more closely related to the element interactions, relationships between element characteristics and goals have been developed. These relationships identify interactions between elements, or changes in a C^2 element, that could impact the ability of a system to meet the criteria required to satisfy a goal.

To function as a complete tool, CMLP needs some capability to measure the effectiveness of a system against hypothetical attack scenarios. The Sensors and Weapons against Threat (SWaT) Model and Capacity Model meet this need.

The SWaT Model utilizes data about the sensors, weapons, and threat that has been input by the user to make an assessment of the number and types of threats that may be killed and the number of weapons used. The assessment is based on probability of the system tracking threats, the number and

types of threats, the number of weapons, and the probability of kill of a weapon versus a specific threat. Additionally, the SWaT model provides for the establishment of zones of defense, which are generalized to be concentric circles known as *range bins*, around a hypothetical defended area. In each of the range bins the user can specify the order in which the threats are to be engaged, the percentage of threats to be engaged, the priority of weapon usage, the percentage of available weapons, and the specific weapons to be used against each threat. The CMLP system uses this data and the probability data supplied about each of the sensors and weapons to compute the number of threats killed and the weapons used for each range bin. The results of each successive range bin are then iteratively fed into the next to perform the same type of calculation. The flexibility provided by the user inputs and the multiple range bins provides the capability to perform "what if" experiments to determine potential strategies to counter a specific threat. The SWaT model is not designed to have the accuracy of the depth a simulation provides, but, assuming realistic data is used as inputs to the probabilistic model, reasonable results can be expected and can assist in pre-simulation analysis to better define simulation experiments to be conducted, thus reducing the amount of simulation time necessary.

The Capacity Model provides the capability to establish an estimate of the various system processing loads (such as speed, memory, and timing) based on the individual loads associated with the performance of the C^2 function groups. It also provides the capability to evaluate the potential processing load impacts as a result of changes in the sensors, weapons, or threats, as well as processing load changes resulting from changes in the functional performance

capability of the C^2 elements. The Capacity Model utilizes an estimate of the percentage of the total current processing capacity (typically in MIPs, bytes, or seconds) that the load represents, and the required spare capacity of the system, as a starting point. The functional groups are then assigned a portion of the total processing load as a percentage, thus creating a point of reference for estimating impacts. The impacts are estimated for each functional group in terms of the percentage change to the numbers of sensors, weapons, or threats and predefined mathematical relationships. These mathematical relationships are in form of linear, square, cubed, logarithmic, or other prespecified expressions applied in any combination to the percentage change in sensors, weapons, and threats. The CMLP system computes new percentages of the processing load based on these mathematical relationships, normalizes the load to 100%, and presents the user with the percentage change for each function group. The user has the capability to edit the new percentage load to account for any other changes that cannot be handled mathematically. As in the case of the SWaT Model, the Capacity Model is not designed to provide exact answers to processing load changes, but to give the user an estimate of the impact of proposed system changes.

A search capability for identifying systems for which particular criteria are met will be provided in a future version of CMLP. This feature, known as Analogy Development, permits the user to find equivalent systems to the one under development and compare them to determine how other systems met a particular goal. Through this means, knowledge gained through the development of one system can be applied to other systems to solve similar problems and to meet the established goals.

B3. OPERATIONAL OVERVIEW

A *baseline* C^2 system stored in the CMLP system is used as a starting point from which new baselines are created and evaluated. These subsequent baselines are called *excursions* during the process of their derivation from the parent baseline. An excursion is created by loading a baseline and altering the information in it relating to any of the system elements, models, or evaluations. Once the user is satisfied with the changes made to the excursion, it is saved and becomes a *new baseline*. The capability of developing new baselines from scratch is available to the user as well. In this case, the same means to enter the data are used, but there are no preexisting data to help the user get started.

All interactions with a baseline are under the control of the user through the *forms* that are generated by the CMLP system. Operations can be controlled through the use of the mouse; the keyboard is used for text entry. The predefined forms guide the user's interactions and provide places to receive text; they display user-entered data and system-generated results. Forms are composed of *boxes* which in turn contain *fields*. In the next section, details of each of the fields, boxes, and *applications* (operations) which comprise the forms in the CMLP system will be presented.

In general, the system is controlled by *selecting* either data appearing in a field within a box or an *application button* by *clicking* the mouse. An application button is a bordered and labeled space within a box which corresponds functionally to a pushbutton on a control panel. Clicking is the process of pointing at a button or data item with the cursor by moving the mouse on the table top, then pressing the *left* button on the mouse and immediately releasing it. The selection of an item or a button will cause the system to take appropriate action.

In the case of data items that are selected, the system may respond by displaying related data in other boxes. Or, the system may take the selected item as the focus of action specified by the subsequent selection of a application button, which often involves adding, deleting, or changing the selected data item (and, potentially, the data in other boxes associated with it). When the system requires that the user supply a value for a new or changed data item, it will either provide a menu for selecting an option (by clicking on it with the mouse) or a box in which a value can be typed. The value will then be added to the appropriate place(s) in the form(s). At times, the user may have the choice of menu selection or entering a new value (previously unknown to the system).

B4. FORM INTERACTIONS

This section presents the forms that comprise the CMLP system. The forms are divided into system-level operations, which allow the user to describe to the CMLP system how it is to manage or make inferences about baseline systems, and baseline-level operations, which allow the user to describe the baselines themselves.

B4.1 SYSTEM-LEVEL OPERATIONS

The system-level operations are conducted through the following forms: goal developer, sensitivity rule developer, function developer, and characteristics developer.

B4.1.1 GOAL REVIEWER/DEVELOPER.

This form (Figure B-1) allows the user access to descriptions of the goals by which the baseline will be assessed (via the goal evaluation form). In reviewer mode, they are available for review only. In developer mode, they may be altered to suit the specific requirements of a particular C^2 system.

- **Standard Applications Box.** Clicking the mouse on the following buttons in the standard applications box has the specified results.

HELP. Invokes the help system if it is not already invoked. Displays the text of the relevant section of this User Manual as the user moves the cursor into a box (except, for technical reasons, a box in which a textual description or textual remarks are displayed). A second click on the button turns this capability off; meanwhile the button remains inverted.

DONE. Removes the form from the screen, and, in developer mode, reports STOREd changes to the knowledge server. Changes that are not STOREd are lost.

CANCEL. Removes the form from the screen. Even STOREd changes are lost.

LOAD. Allows the user to select from the SAVEd goal knowledge bases.

SAVE. In developer mode, allows the user to save the current state of the goal knowledge base for LOADing in a later session. If the user does not SAVE the knowledge base, but leaves the form via DONE, any changes will

be reported to the knowledge server and will remain in force until the user EXITs from CMLP. The changed data will be available for subsequent invocations of the goal developer within the current session; the SAVE need only be done before session EXIT.

- **Goal Group Box.** The goals are organized into goal groups, through which they are selected for display in the form. The goal group box lists the available goal groups. Clicking the mouse on the desired goal group will result in the names of the goals in the group being displayed in the goal name box.

- **Goal Name Box.** This box allows the user to specify the goal to be displayed on the form. Clicking the mouse on an existing goal will result in the display of the data for that goal. This box is used in conjunction with the goal developer application buttons box.

- **Goal Developer Application Buttons Box.** The application buttons allow the user to add and delete goals as well as to save the resultant changes (developer mode only). Another allows the user to. deselect the currently selected goal. The buttons are described below.

STORE. In developer mode, saves the changes made during interaction with the form.

REVERT. In developer mode, discards any changes made since the last time the STORE button was pressed, or the opening of the form if it has not yet been pressed.

NEW. In developer mode, allows the creation of a new goal within the selected goal group. Clicking the mouse button on NEW will result in the display being cleared so that the data for it can be added by the user. If no goal group is selected (see DESELECT below), adds a new goal group.

DELETE. In developer mode, deletes the currently selected goal and its associated data. If there are no goals in the selected goal group, deletes the goal group and associated data.

DESELECT. Removes the current selection(s).

```
┌─────────────────────────────────────────────────────────────────────────┐
│ GOAL DEVELOPER                                                            │
│ ┌───────┐ ┌───────┐ ┌───────┐ ┌───────┐ ┌───────┐                         │
│ │ HELP  │ │ DONE  │ │CANCEL │ │ LOAD  │ │ SAVE  │                         │
│ └───────┘ └───────┘ └───────┘ └───────┘ └───────┘                         │
```

GOAL GROUP

System Robustness Goals
Surveillance Data Use Goals
System Response Selection Goal
System Timeliness Goals
System Authority Control Goals
System Survivability Goals

GENERAL GOAL

CONSIDER ASSETS AND RISK
OPTIMIZE TIME TO INTERCEPT
MULTIPLE WEAPONS COMMITMNTS
ENVIRNMNTL & GEOMETRC CONDS
GIVE UPDATES TO WEAPONS

STORE
REVERT
NEW
DELETE
DESELECT

GOAL DESCRIPTION

The C2 System shall consider the value of assets
at risk, degree of risk, probability of defensive
actions success, and the risk to assets that may
result from defensive actions.

☐ Satisfactory

☑ Partially Satisfactory

☐ Unsatisfactory

☐ Not Supported

☐ Not Applicable

CRITERION FOR DEGREE OF GOAL SATISFACTION

PARTIALLY SATISFACTORY - The system utilizes
limited information and rules of thumb for
assigning weapons to specific threats or makes
unsupported assumptions in the nature of the threat
type for performing assignment.

AFFECTS
Swel Model ◌ Yes Capacity Model ◌ No Geography Model ◌ No

APPLICABLE FUNCTIONS

Raid Composition
Strength
Priority Ranking
Weapon Probability of Kill
Weapon-Target Assignment

ADD
DETAILS
DELETE

ADDITIONAL REMARKS

89-05-014

Figure B-1. Goal Reviewer/Developer Form

• **Goal Description Box.** This box is displayed below the above three boxes and displays the description of the goal.

• **Criterion Selection Box.** The goal criterion selection box allows the user to select the degree of satisfaction criterion displayed in the satisfaction criterion box to its right. Clicking on a selection displays the text of the criterion.

• **Satisfaction Criterion Box.** This box describes the conditions that must be met for the selected goal to be satisfied to the selected degree.

• **Affects Box.** This box shows whether other specified component models can affect the selected goal's evaluation (i.e., judged degree of goal satisfaction).

- *Applicable Functions Box.* This box lists functions for which a status change can affect the selected goal's evaluation. Clicking on a function selects it for action by the applicable functions buttons.

- *Applicable Functions Buttons.* These buttons act as follows:

ADD. Allows the user to add a function to the list of applicable functions. Pops up a list of functions that are not currently in the list and allows the user to select one of them.

DETAILS. Displays the function developer form initialized to the selected applicable function, or initializes the function developer form to it if it is already on screen.

DELETE. Deletes the selected function from the list of applicable functions.

- *Additional Remarks Box.* Displays textual additional remarks about the goal. In developer mode, allows the user to STORE and SAVE textually entered additional remarks.

B4.1.2 SYSTEM SENSITIVITY RULE DEVELOPER. The system sensitivity rule reviewer/developer (Figure B-2) allows the user to review relationships between system elements and characteristics. In developer mode, the user can modify them to meet the specific needs of any class of C^2 system the user may wish to consider.

- *Standard Applications Box.* Clicking the mouse on the following buttons in the standard applications box has the specified results.

HELP. Invokes the help system if it is not already invoked. Displays the text of the relevant section of this User Manual as the user moves the cursor into a box (except, for technical reasons, a box in which a textual description or textual remarks are displayed). A second click on the button turns this capability off; meanwhile the button remains inverted.

DONE. Removes the form from the screen, and, in developer mode, reports STOREd changes to the knowledge server. Changes that are not STOREd are lost.

CANCEL. Removes the form from the screen. Even STOREd changes are lost.

LOAD. Allows the user to select from the saved rule knowledge bases.

SAVE. In developer mode, allows the user to save the current state of the rule knowledge base for LOADing in a later session. If the user does not SAVE the knowledge base, but leaves the form via DONE, any changes will be reported to the knowledge server and will remain in force until the user EXITs from CMLP. The changed data will be available for subsequent invocations of the rule developer within the current session; the SAVE need only be done before session EXIT.

- *Excursion Element Box.* This box allows the user to select the element of the system for which changes in characteristics will be entered. The elements are sensors, weapons, threats, and C^2. Clicking the mouse button on the desired name will result in the existing characteristics of the item being displayed in the excursion characteristics box.

- *Excursion Characteristics Box.* This box provides for the selection of the specific characteristic of the selected element.

- *Excursion Change Box.* This box provides for the specification of the type of change in a characteristic previously selected.

- *Impact Element Box.* This box allows the user to select the element of the system on which a characteristic change will have an effect. Clicking the mouse button on the desired name will result in the impacted characteristics of the item being displayed in the impact characteristics box.

- *Impact Characteristics Box.* This box allows the selection of the characteristic on which the selected changes will have an effect.

- *Impact Change Box.* This box allows the selection of the type of change that will occur in the impact characteristic as a result of a selected excursion on a characteristic.

- *Rule Developer Buttons Box.* In developer mode, the rule developer buttons allow the user to save new rules or delete existing rules. The buttons act as described below.

Figure B-2. SS Rule Reviewer/Developer Form

STORE. In developer mode, saves the changes made during interaction with the form.

REVERT. In developer mode, discards any changes made since the last time the STORE button was pressed, or the opening of the form if it has not yet been pressed.

NEW. In developer mode, allows the creation of a new rule. Clicking the mouse button on NEW will result in the display being cleared so that the data for it can be added by the user.

DELETE. In developer mode, deletes the currently selected goal and its associated

data. If there are no goals in the selected goal group, deletes the goal group and associated data.

• **Impact Buttons Box.** This box allows the user to search the rulebase for system-subsystem interactions. The search can be either "forwards" (specify an excursion to find impacts) or "backwards" (specify an impact to see what excursions could impact it).

FIND. Select one or more of excursion element, excursion characteristic, impact element, and impact characteristic, then click on FIND. If there is a rule which matches the specified search conditions, it will be displayed. If there is a further rule, the FIND button will remain inverted to signify that. Press FIND again to display that rule (and so on). When there are no more rules that match, the inversion is removed. To abort after the display of one or more matching rules, click on DESELECT.

DESELECT. Deselects the current rule, displaying only excursion and impact elements. If in the middle of a FIND, aborts the display of remaining matching rules.

• **Rationale Box.** This box displays the rationale for the displayed rule. In developer mode, provides for the entry or update of a rationale.

• **Notes Box.** This box displays the notes for the displayed rule. In developer mode, provides for the entry or update of notes.

• **Applicable Goals Box.** This box permits the user to select the goals that may be impacted as a result of selecting a qualitative excursion in a baseline, or displays applicable goals from a selected system sensitivity rule.

• **Applicable Goals Application Buttons.** These buttons provide the capability to include selected goals in a system sensitivity rule. The buttons are as described below.

ADD. Allows the user to add a goal to the list of applicable goals. Pops up a list of goals that are not currently in the list and allows the user to select one of them.

DETAILS. Displays the goal developer form initialized to the selected applicable goal, or initializes the goal developer form to it if it is already on screen.

DELETE. Deletes the selected goal from the list of applicable goals.

B4.1.3 C² FUNCTION REVIEWER/ DEVELOPER. The function reviewer/ developer (Figure B-3) allows the user to review the functions of the various elements. In developer mode, it allows the user to create and define functions necessary to fit the

Figure B-3. C² Function Reviewer/Developer Form

specific needs of the system under development.

• **Standard Applications Box.** Clicking the mouse on the following buttons in the standard applications box has the specified results.

HELP. Invokes the help system if it is not already invoked. Displays the text of the relevant section of this User Manual as the user moves the cursor into a box (except, for technical reasons, a box in which a textual description or textual remarks are displayed). A second click on the button turns this capability off; meanwhile the button remains inverted.

DONE. Removes the form from the screen, and, in developer mode, reports STOREd changes to the knowledge server. Changes that are not STOREd are lost.

CANCEL. Removes the form from the screen. Even STOREd changes are lost.

LOAD. Allows the user to select from the saved function knowledge bases.

SAVE. In developer mode, allows the user to save the current state of the function knowledge base for LOADing in a later session. If the user does not SAVE the knowledge base, but leaves the form via DONE, any changes will be reported to the knowledge server and will remain in force until the user EXITs from CMLP. The changed data will be available for subsequent invocations of the function developer within the current session; the SAVE need only be done before session EXIT.

• **Function Group Box.** This box permits the user to select a C^2 function group. When the user clicks the mouse on a function group, its function will be presented in the function box.

• **Function Box.** The functions for the selected function group are displayed in the function box. A scroll bar is provided so that all functions can be reviewed. Functions may be added or deleted through the use of the function application buttons. When the user clicks on a function, the description is displayed in the function description box.

• **Function Application Buttons Box.** The function application buttons are provided to edit the functions for the selected function group. The buttons are described below.

STORE. In developer mode, saves the changes made during interaction with the form.

REVERT. In developer mode, discards any changes made since the last time the STORE button was pressed, or the opening of the form if it has not yet been pressed.

NEW. In developer mode, allows the creation of a new function within the selected function group. Clicking the mouse button on NEW will result in the display being cleared so that the data for it can be added by the user. If no function group is selected (see DESELECT below), adds a new function group.

DELETE. In developer mode, deletes the currently selected function and its associated data. If there are no functions in the selected function group, deletes the function group and associated data.

DESELECT. Removes the current selection(s).

• **Function Description Box.** This box displays the description of a selected function, or permits the entry of a description for a new function.

B4.1.4 CHARACTERISTIC DEVELOPER. The characteristic developer (Figure B-4) allows the user to define the characteristics of the C^2 system under development.

• **Standard Applications Box.** Clicking the mouse on the following buttons in the standard applications box has the specified results.

HELP. Invokes the help system if it is not already invoked. Displays the text of the relevant section of this User Manual as the user moves the cursor into a box (except, for technical reasons, a box in which a textual description or textual remarks are displayed). A second click on the button turns this capability off; meanwhile the button remains inverted.

Figure B-4. Characteristic Developer

DONE. Removes the form from the screen, and, in developer mode, reports STOREd changes to the knowledge server. Changes that are not STOREd are lost.

CANCEL. Removes the form from the screen. Even STOREd changes are lost.

LOAD. Allows the user to select from the saved function knowledge bases.

SAVE. In developer mode, allows the user to save the current state of the characteristics knowledge base for LOADing in a later session. If the user does not SAVE the knowledge base, but leaves the form via DONE, any changes will be reported to the knowledge server and will remain in force until the user EXITs from CMLP. The changed data will be available for subsequent invocations of the function developer within the current session; the SAVE need only be done before session EXIT.

• *Element Box.* This box permits the user to select an element by clicking the mouse on the desired element. The list of characteristics will be presented in the characteristic name box.

• *Characteristic Box.* The characteristics are presented for the selected element. A scroll bar is provided to permit review of all of the names. Characteristics can be added or deleted through the use of the characteristic application buttons. Clicking on a character-istic causes its description to be displayed in the characteristic description box.

• *Characteristic Application Buttons Box.* The characteristic application buttons allow the user to edit the characteristics of a selected element. The buttons are described below.

STORE. In developer mode, saves the changes made during interaction with the form.

REVERT. In developer mode, discards any changes made since the last time the STORE button was pressed, or the opening of the form if it has not yet been pressed.

NEW. In developer mode, allows the creation of a new characteristic within the selected element. Clicking the mouse button on NEW will result in the display being cleared so that the data for it can be added by the user.

DELETE. In developer mode, deletes the currently selected characteristic and its associated data.

DESELECT. Removes the current selection(s).

• *Characteristic Description Box.* This box displays the description of a selected characteristic. In developer mode, permits the entry of a description for a new characteristic.

B4.2 BASELINE-LEVEL OPERATIONS

Baseline-level operations are accomplished through the following forms: baseline summary display, C^2 element specification, sensor/range specification, weapons specification, and threat specification.

B4.2.1 BASELINE SUMMARY DISPLAY.

The summary display (Figure B-5) allows the user to load a baseline system from a set of available baselines, review top-level summaries of its major elements, and obtain access to more detailed levels.

• **Standard Applications Box.** Clicking the mouse on the following buttons in the standard applications box has the specified results.

HELP. Invokes the help system if it is not already invoked. Displays the text of the relevant section of this User Manual as the user moves the cursor into a box (except, for

Figure B-5. Baseline Summary Display Form

technical reasons, a box in which a textual description or textual remarks are displayed). A second click on the button turns this capability off; meanwhile the button remains inverted.

DONE. Removes the form from the screen, and, in developer mode, reports STOREd changes to the knowledge server. Changes that are not STOREd are lost.

CANCEL. Removes the form from the screen. Even STOREd changes are lost.

LOAD. Allows the user to select from the saved baseline knowledge bases.

SAVE. In developer mode, allows the user to save the current state of the baseline knowledge base for LOADing in a later session. If the user does not SAVE the knowledge base, but leaves the form via DONE, any changes will be reported to the knowledge server and will remain in force until the user EXITs from CMLP. The changed data will be available for subsequent invocations of the baseline developer within the current session; the SAVE need only be done before session EXIT.

• **Version Data Box.** This box contains general information about the baseline being displayed. The data include the following:

Author. The user responsible for creation of the displayed version of the baseline.

Date. The date the displayed baseline version was created.

System Type. The type of C^2 system represented by the baseline (ADI, SDI, SPADOCC, or other user defined type).

Parent System Type. The system type of the baseline from which the displayed version was created.

Parent Number. The version number of the baseline from which the current baseline was developed.

• **C^2 Element Box.** This box contains the type, level, and number of each C^2 element stored as part of the baseline. The C^2 elements will be listed in descending level order. The specific categories are:

C^2 Element. The name of the C^2 element.

Level. The level of the C^2 element type in this baseline (Level = 1 represents the highest command authority).

Number. The number of this type of C^2 element in this baseline.

• **Goal Evaluation Summary Box.** This box contains an overview by goal group of the overall evaluation of the baseline. The categories are defined below.

General Goal Group. The name of each of the goal groups.

Total. The number of goals in the group.

Rating Columns. The number of goals in the group in each rating category. The categories are: Satisfied, Partially Satisfied, Unsatisfied, Not Supported, and Not Applicable.

• **Sensors Box.** This box provides a summary of each of the unique sensor types used in the baseline and their respective number. An entry is provided for each type in each range bin.

• **Weapon Platforms Box.** This box provides a summary of each of the unique weapon platform types used in the baseline and their respective number. An entry is provided for each type in each range bin.

• **Threat Carriers Box.** This box provides a summary of each of the unique threat carrier types used in the baseline and their respective number.

• **Comment Box.** This box shows the notes stored with the currently displayed baseline. It is read-only; changes to it are never saved.

• **Notes Box.** This box allows the user to comment on the current baseline, as developed via excursions. It will be saved as a comment field when the development baseline is saved.

• **Excursion Box.** This box shows excursions that have been made to the development baseline.

B4.2.2 EXCURSION FORM. The excursion form (Figure B-6) allows the user to see

```
EXCURSION REVIEWER
   HELP        DONE
 ELEMENT           ASPECT        ORIGINAL   CURRENT          DETAILS
 ----------------  ------------  ---------  ---------        IMPACT
 Bomber            Number               80        180
 EOB Maintenance   Func Stat     Manual     Automatic
 Wpn-Target Assgnmnt  Func Stat  Not Done   Manual
 Display Generation  Func Stat   Manual     Auto & Ma
 ----------------  ------------  ---------  ---------
```

Figure B-6. Excursion Analysis Form

the excursions that have been applied to the development baseline.

• *Standard Applications Box.* Clicking the mouse on the following buttons in the standard applications box has the specified results.

HELP. Invokes the help system if it is not already invoked. Displays the text of the relevant section of this User Manual as the user moves the cursor into a box (except, for technical reasons, a box in which a textual description or textual remarks are displayed). A second click on the button turns this capability off; meanwhile the button remains inverted.

DONE. Removes the form from the screen.

• *Excursion Box.* This box permits the user to review the excursions. The fields are described below.

Element. Shows the system element that was changed.

Aspect. Shows the attribute that was changed.

Original. The evaluation before it was changed.

Current. The current evaluation.

• *Excursion Application Buttons Box.* The buttons are described below.

DETAILS. Displays the current state of the selected element.

IMPACT. Displays the evaluation assessment form appropriate to the element.

B4.2.3 C² ELEMENT SPECIFICATION FORM. This form (Figure B-7) allows the user to define specific command elements, their

hierarchy, and their functions and characteristics.

• *Standard Applications Box.* Clicking the mouse on the following buttons in the standard applications box has the specified results.

HELP. Invokes the help system if it is not already invoked. Displays the text of the relevant section of this User Manual as the user moves the cursor into a box (except, for technical reasons, a box in which a textual description or textual remarks are displayed). A second click on the button turns this capability off; meanwhile the button remains inverted.

DONE. Removes the form from the screen, and, in developer mode, reports STOREd changes to the knowledge server. Changes that are not STOREd are lost.

CANCEL. Removes the form from the screen. Even STOREd changes are lost.

• *C² Elements Box.* This box supports the selection of C² element types, as well as their functions, characteristics and descriptions. Using the C² element application buttons, these data can be modified by the user. The following is a description of the fields.

Type. The (user-definable) name of the C² element.

Number. The number of a particular type of C² element in the baseline.

Level. The level of command assigned to the C² element. Level = 1 is the highest command authority; C² elements are ordered in descending order of command authority.

89-05-010

Figure B-7. C² Element Specification Form

• **C² Element Application Buttons.** C² element application buttons permit the user to modify the data pertaining to the type, number, and level of the C² elements, and allow the elicitation of an aggregate list of C² element functions and characteristics for an excursion. The application button functions are described below.

ADD. Allows the addition of a new C² element type to the development baseline. If the desired element type does not yet exist, the user may create a NEW one. If the user does not CANCEL the ADD, the boxes will be cleared so that the data for it can be supplied by the user.

MODIFY. Allows the modification of the Level and/or Number data for the selected C^2 element.

DELETE. Allows the deletion of the selected C^2 element from the development baseline.

STORE. Saves the changes made during interaction with the form.

REVERT. Discards any changes made since the last time the STORE button was pressed, or the opening of the form if it has not yet been pressed.

- *C^2 Functions/Status Box.* This box is the area in which the function groups, functions, and the status of each function are displayed for the selected C^2 element. A scroll bar can be used to view all of the data. The status column to the right of the function name defines the performance level of the function as Automated, Manual, Automated and Manual, Not Done, or Not Supported (by sensor and weapons). These are contracted as required to fit in the column width. Clicking the mouse button on a particular function results in the display of its description field.

- *Function Application Buttons.* The function application buttons provide the capability for the user to add functions that currently have a status of Not Done or Not Supported to modify the status of an existing function. The buttons are described below.

ADD. Allows the user to add a function with the status of Not Done or Not Supported (with a new, active status).

MODIFY. Allows the user to change the status of a selected function. If the status is changed to Not Done or Not Supported the function will be placed in the summary box.

- *C^2 Characteristic/Status Box.* This box displays the C^2 characteristics and their statuses for the entire development baseline. A scroll bar is provided. The status column to the right of the characteristic name defines the performance level of the characteristic as follows:

Yes. The capability exits.

No. This capability does not exit.

Partial. A limited capability exits in this area.

Clicking the mouse on a characteristic results in the display of the description of the characteristic in the description box.

- *Characteristic Application Buttons.* The characteristic application buttons allow the user to modify the status of a characteristic. There is one button:

MODIFY. Allows the user to change the status of a characteristic.

- *Description Box.* This box contains a description of a C^2 element, function, or characteristic selected by the user by clicking with the mouse.

- *Summary Box.* This box displays information regarding functions which have the status of Not Done or Not Supported.

B4.2.4 SENSOR/RANGE SPECIFICATION FORM. The sensor/range form (Figure B-8) allows the user to specify any number of *range bins* (defensive zones), specify the sensors in a range bin, and provide the percentage tracked of individual target types in each specified range bin.

- *Standard Applications Box.* Clicking the mouse on the following buttons in the standard applications box has the specified results.

HELP. Invokes the help system if it is not already invoked. Displays the text of the relevant section of this User Manual as the user moves the cursor into a box (except, for technical reasons, a box in which a textual description or textual remarks are displayed). A second click on the button turns this capability off; meanwhile the button remains inverted.

DONE. Removes the form from the screen, and, in developer mode, reports STOREd changes to the knowledge server. Changes that are not STOREd are lost.

CANCEL. Removes the form from the screen. Even STOREd changes are lost.

- *Range Bin Box.* This box allows the user to select a specific range bin. Selection of a

Figure B-8. Sensor/Range Specification Form

range bin leads to a display of the sensor data for the range bin.

• **Range Bin Buttons Box.** The application buttons function as follows.

NEW. Allows the user to create a new range bin for the baseline.

DELETE. Deletes the selected range bin and all information associated with it and subsequent range bins.

• **Sensor Type/Number Box.** This box permits the user to review and modify the sensor systems in a baseline, and to change their number in each of the user-defined range bins. The categories are defined below.

Sensor Type. Identifies the type of sensor, which may be selected from one of those defined within the system or may be a

new type defined by the user by means of the sensor application buttons.

Number. Defines the number of the particular sensor type in the selected range bin. The number field can be edited using the application buttons.

• **Sensor Type Application Button Box.** This box allows the user to edit the sensor type and sensor number fields. The functions of the buttons are described below.

ADD. Allows the addition of a new sensor type to the development baseline. If the desired sensor type does not yet exist, the user may create a NEW one. If the user does not CANCEL the ADD, the boxes will be cleared so that the data for it can be supplied by the user.

MODIFY. Allows the modification of the Level and/or Number data for the selected C^2 element.

DELETE. Allows the deletion of the selected C^2 element from the development baseline.

STORE. Saves the changes made during interaction with the form.

REVERT. Discards any changes made since the last time the STORE button was pressed, or the opening of the form if it has not yet been pressed.

● *Target Type/Percent Tracked Box.* This box allows the user to review and edit the target types detectable within a range bin and their probability of tracking by the system. The percent tracked field contains the value used in the SWaT model to determine the probability of tracking each target type in the specified range bin. This value may be edited using the application buttons.

● *Target/Percent Tracked Application Buttons.* The target/percent tracked application buttons allow the user to modify the percent tracked of a target by the development baseline at the selected range bin. There is one button:

MODIFY. Allows the user to change the percent tracked of the selected target.

● *Sensor Description Box.* This box displays the description of a selected sensor type, or permits the entry of a description of a new sensor type. It also displays the definition of the selected target.

B4.2.5 WEAPONS SPECIFICATION FORM. The weapons specification form (Figure B-9) allows the user to create and define weapon types for use in the SWaT model and the impact analysis functions.

● *Standard Applications Box.* Clicking the mouse on the following buttons in the standard applications box has the specified results.

HELP. Invokes the help system if it is not already invoked. Displays the text of the relevant section of this User Manual as the user moves the cursor into a box (except, for technical reasons, a box in which a textual

description or textual remarks are displayed). A second click on the button turns this capability off; meanwhile the button remains inverted.

DONE. Removes the form from the screen, and, in developer mode, reports STOREd changes to the knowledge server. Changes that are not STOREd are lost.

CANCEL. Removes the form from the screen. Even STOREd changes are lost.

● *Range Box.* This box allows the user to select a specific range bin for the purpose of reviewing or editing the weapon platform data in the selected range bin. Selecting a range bin causes the weapon platforms specified in it to be displayed in the weapon platform box.

● *Weapon Platform Box.* This box allows the user to review or modify the weapon platform data for the selected range bin. Clicking the mouse on a platform causes data about it to be displayed in the armament and description boxes. The fields in the weapon platform box are described below.

Weapon Platform Type. identifies the type of weapon platform. The type can be selected from one of those defined within the system, or a new type can be defined by the user with the application buttons.

Number. Defines the total number of the particular weapon platform in a selected range bin. The number field may be modified by the user with the application buttons.

● *Weapon Platform Application Buttons Box.* This box allows the user to edit the weapon platform type and number fields. The functions of the buttons are described below.

ADD. Allows the addition of a new weapon platform type to the development baseline. If the desired weapon platform type does not yet exist, the user may create a NEW one. If the user does not CANCEL the ADD, the boxes will be cleared so that the data for it can be supplied by the user.

MODIFY. Allows the modification of the Number data for the selected weapon platform.

```
WEAPON SPECIFICATION

[ HELP ]    [ DONE ]    [ CANCEL ]

Range Switch
RANGE  ○ 1

WEAPON PLATFORM TYPE    NUMBER          [ ADD ]
----------------------  ------          [ MODIFY ]
Interceptor               300           [ DELETE ]
Patriot Launcher           50           [ STORE ]
----------------------  ------          [ REVERT ]


ARMAMENT                NUMBER          [ ADD ]
----------------------  ------          [ MODIFY ]
Patriot                    6            [ DELETE ]
----------------------  ------


TARGET                  %(KILL)         [ MODIFY ]
----------------------  ------
Bomber                    95
High Fast Cruise Missile  80
Low Slow Cruise Missile   40
Cruise Missile Carrier    10
----------------------  ------

Patriot SAM:
    A medium range surface to air missile for
 air
defense.  The Patriot missile is the successo
r to the HAWK
Missile.
```

Figure B-9. Weapons Specification Form

DELETE. Allows the deletion of the selected weapon platform from the development baseline.

STORE. Saves the changes made during interaction with the form.

REVERT. Discards any changes made since the last time the STORE button was pressed, or the opening of the form if it has not yet been pressed.

• **Armament Type Box.** This box allows the user to specify the configuration of the weapons platform in terms of the armament it carries. Clicking the mouse on a specific armament type will cause a display of the target types in the range bin, the percent killed with this armament. The fields are described below.

Armament. Identifies the specific weapons carried by the selected weapon platform above. This field can be modified by means of the application buttons.

Number. Contains the number of the specific armament on the selected weapon platform. This field can be modified using the application buttons.

• **Armament Application Buttons Box.** This box allows the user to edit the armament type and number fields. The functions of the buttons are described below.

ADD. Allows the addition of a new weapon platform type to the development baseline. If the desired weapon platform type does not yet exist, the user may create a NEW one. If the user does not CANCEL the ADD, the boxes will be cleared so that the data for it can be supplied by the user.

MODIFY. Allows the modification of the Number data for the selected weapon platform.

DELETE. Allows the deletion of the selected weapon platform from the development baseline.

• **Target Percent Killed Box.** This box allows the user to specify the probability of kill for a selected armament type, and whether the armament type should be used on a particular target. The fields are described below.

Target Type. Identifies each of the targets that have been entered by the user in the threat form. This field is not editable in this box.

%(Kill). Provides the selected armament's probability of kill (used in the SWaT model) against each of the target types.

• **Weapon/Armament Description Box.** This box permits the user to review the description of a selected weapon or armament, or to enter a description of a new weapon or armament type.

B4.2.6 THREAT SPECIFICATION FORM. The threat specification form (Figure B-10) allows the user to create and define threat types for use in the SWaT model and the impact analysis functions.

• **Standard Applications Box.** Clicking the mouse on the following buttons in the standard applications box has the specified results.

HELP. Invokes the help system if it is not already invoked. Displays the text of the relevant section of this User Manual as the user moves the cursor into a box (except, for technical reasons, a box in which a textual

Figure B-10. Threat Specification Form

description or textual remarks are displayed). A second click on the button turns this capability off; meanwhile the button remains inverted.

DONE. Removes the form from the screen, and, in developer mode, reports STOREd changes to the knowledge server. Changes that are not STOREd are lost.

CANCEL. Removes the form from the screen. Even STOREd changes are lost.

• **Threat Carrier Box.** This box allows the user to review and edit the threat carriers in the baseline. Selection of a threat carrier type by clicking the mouse on the item will display the threats carried by the threat carrier, the range at which they will be deployed, and a description of the threat (upon request). The fields of the threat carrier box are described below.

Threat Carrier Type. Identifies the threat carriers. The threat types field can be edited using the application buttons.

Number. Contains the number of the threat type. The number field can be modified using the application buttons.

• **Threat Carrier Type Application Button Box.** This box allows the user to edit the data associated with the threat platforms and their threats. The functions of the buttons are described below.

ADD. Allows the addition of a new threat carrier type to the development baseline. If the desired threat carrier type does not yet exist, the user may create a NEW one. If the user does not CANCEL the ADD, the boxes will be cleared so that the data for it can be supplied by the user.

MODIFY. Allows the modification of the Number data for the selected threat carrier.

DELETE. Allows the deletion of the selected threat carrier from the development baseline.

STORE. Saves the changes made during interaction with the form.

REVERT. Discards any changes made since the last time the STORE button was pressed, or the opening of the form if it has not yet been pressed.

• **Threat Type Box.** This box specifies the type of threats carried by each threat platform and the range at which they will be deployed. The fields are described below.

Threat Type. Identifies the threats carried by the selected threat carrier. This field can be modified using the application buttons.

Deployment Range. The range at which all or a portion of a specific threat may be deployed from a selected threat carrier.

Number Deployed at Range. The number of the threat type deployed at the range specified in the deployment range field. This field can be modified using the application buttons.

• **Threat Type Application Buttons Box.** This box permits the user to edit the data associated with the threats. The functions of the buttons are described below.

ADD. Allows the addition of a new threat carrier type to the development baseline. If the desired threat carrier type does not yet exist, the user may create a NEW one. If the user does not CANCEL the ADD, the boxes will be cleared so that the data for it can be supplied by the user.

MODIFY. Allows the modification of the Number data for the selected threat carrier.

DELETE. Allows the deletion of the selected threat carrier from the development baseline.

• **Threat Platform/Threat Description Box.** This box allows the user to review the description of a selected threat platform or threat, or to enter a description of a new threat platform or threat type.

B4.2.7 IMPACT ANALYSIS FORM. This form is the same as the excursion form.

B4.2.8 CAPACITY EVALUATION FORM. This form (Figure B-11) allows the user to generate an estimate of the required processing load capacity of the C^2 system and to examine the impact on processing load of hypothetical changes in the system.

• **Standard Applications Box.** Clicking the mouse on the following buttons in the

```
┌─────────────────────────────────────────────────────────────┐
│ CAPACITY EVALUATION                                          │
│  ┌───────┐ ┌───────┐ ┌─────────┐                            │
│  │ HELP  │ │ DONE  │ │ CANCEL  │                            │
│  └───────┘ └───────┘ └─────────┘                            │
│ ┌──────────────────────────────────┬────────────────────────┤
│ │ PROCESSING CAPACITY      Percent │  ┌─────────┐           │
│ │ ────────────────────────  ────── │  │ MODIFY  │           │
│ │                                  │  └─────────┘           │
│ │ Design Load                  55  │  ┌─────────┐           │
│ │ Spare Requirement            25  │  │ REVERT  │           │
│ │ ────────────────────────  ────── │  └─────────┘           │
│ │                                  │  ┌─────────┐           │
│ │                                  │  │ STORE   │           │
│ │                                  │  └─────────┘           │
```

Figure B-11. Capacity Analysis Form

standard applications box has the specified results.

HELP. Invokes the help system if it is not already invoked. Displays the text of the relevant section of this User Manual as the user moves the cursor into a box (except, for technical reasons, a box in which a textual description or textual remarks are displayed). A second click on the button turns this capability off; meanwhile the button remains inverted.

DONE. Removes the form from the screen, and, in developer mode, reports STOREd changes to the knowledge server. Changes that are not STOREd are lost.

CANCEL. Removes the form from the screen. Even STOREd changes are lost.

• *Processing Load Design Display Box.* This box contains general information regarding the estimated processing required for the baseline system, the processing load as a percentage of the total estimated processing capacity, and the percentage spare processing capacity required of the system. The fields are described below.

Processing Capacity. Displays the category: Design Load or Spare Requirement.

Percent. Shows the processing capacity requirement.

• *Processing Load Design Buttons.* This box allows the user to edit the data associated with the threat platforms and their threats. The functions of the buttons are described below.

MODIFY. Allows the modification of the percentage data for the selected category.

STORE. Saves the changes made during interaction with the form.

REVERT. Discards any changes made since the last time the STORE button was pressed, or the opening of the form if it has not yet been pressed.

- **Capacity Evaluation Box.** This box presents the user with a display of the processing load breakdown for each of the function groups. The fields are described below.

Function Group. The C^2 function groups that have defined relationships between the changes and their impact on the processing. Selection of a function group by clicking the mouse on the item will result in the mathematical equation for computing percentage change being displayed in the relationship box.

Requirement. The percent processing required by the function group. ·

- **Capacity Evaluation Buttons.** The capacity evaluation application buttons box allows the user to edit the requirement. There is one button:

MODIFY. Allows the modification of the percentage data for the selected function group.

- **Relationship Box.** This box supports the review and modification of the mathematical equation used to compute the function load. The relations are defined in terms of the percentage change to sensors, weapons, and threat. The percentage change to sensors, weapons, and threat is arrived at by computing the net change in terms of a percentage in the numbers of these items from the baseline to those used in the current SWaT model. The user has the capability to edit and modify the equation as necessary to accommodate any special requirements or anomalies in the processing that may occur due to changes in the function performance or the algorithms used for the functions.

- **Relationship Application Buttons.** The relationship application buttons provide the capability for the user to modify the mathematical expressions that define the method the capacity model uses to define the new percentage for a given function. There is one button.

MODIFY. Allows the user to modify the expression governing the new percent computation of a selected function.

- **Comments Box.** This box supports the entry of textual messages that will be saved by the system.

- **Capacity Evaluation Usage Instructions.** The user selects the capacity evaluation from the main menu bar and is presented with the capacity display form initialized with information from the baseline. Before performing the capacity evaluation all system change data, impacts, SWaT, and goal evaluation should be completed owing to potential effects on capacity. A typical sequence of steps for utilizing the capacity evaluation is as follows.

1. Observe the excursion changes box on baseline summary display for changes that have been made to the baseline.

2. Observe the capacity evaluation box and note that the new % column has changed based upon sensor, weapon, or threat changes. These changes that are performed automatically based on the % change in the total number of sensors, weapons, or threat for the system.

3. To account for qualitative changes such as change in functional performance or a characteristic, select the function group and, using the MODIFY button, edit the percentage field for the new value. Repeat this for all desired function groups.

4. To alter a relationship, move to the relationship box, select the function group, and select the MODIFY button. Select the affecting variable (sensors, weapons, or threat) and select the desired mathematical relation (linear, squared, cubed, log, etc.). Perform this for all desired affecting variables for a single function group, then select STORE.

5. Perform Step 4, as desired, for all other function groups.

6. When all function group values have been manipulated as desired, observe the percentage of total processing value and the spare capacity value.

B4.2.9 SWaT ANALYSIS FORM. This form (Figure B-12) allows the user to determine the probabilistic results of a force-on-force exchange based on user-supplied parameters and weapon utilization strategies. The SWaT model provides for multiple defense zones (range bins) and supports iterative trials to achieve a desired outcome.

- **Standard Applications Box.** Clicking the mouse on the following buttons in the standard applications box has the specified results.

    HELP. Invokes the help system if it is not already invoked. Displays the text of the relevant section of this User Manual as the user moves the cursor into a box (except, for technical reasons, a box in which a textual description or textual remarks are displayed). A second click on the button turns this capability off; meanwhile the button remains inverted.

    DONE. Removes the form from the screen, and, in developer mode, reports STOREd changes to the knowledge server. Changes that are not STOREd are lost.

    CANCEL. Removes the form from the screen. Even STOREd changes are lost.

    PRINT. Prints all the contents of all the browsers at every range for the current trial as "swat.print" in the current directory.

- **Range Selection Box.** This box allows the user to review and modify the target, weapon platform, and armament data for the selected range. It also allows the user to select the tactic for the range.

Range 1 Results

Range 2 Results

Figure B-12. SWaT Analysis Form

Range. The range bin of interest for calculating the SWaT solution. The range bins are assigned through the sensors display and are not modifiable from the SWaT display.

Tactic. Allows the user to select between single shots on target or shoot-look-shoot, permitting a maximum of two shots on each target.

• **Trial Box.** This box allows the user to recompute different versions of the same range bin and to step through them to compare results. Once the STORE button is selected (see below), only the version displayed will be saved for further SWaT calculations and all other versions will be discarded. Selection of a trial fills the boxes with its data.

• **SWaT Application Buttons.** These buttons permit the user to calculate and store the results of the SWaT model. The buttons are described below.

CALC. Calculates the SWaT results based on the current parameter settings.

NEW. Creates a new trial.

STORE. Deletes all but the current trial, which becomes trial 1.

• **Target Box.** This box supports the review of target information for a selected range bin, specifies the priority in which targets will be engaged, and provides the expected percentage of targets that will be engageable. The fields are described below.

Target. The target type as specified from the threat platform display for the selected range bin.

Number. The number of each of the target types for the selected range bin as specified in the threat platform display.

Priority. The priority in which the targets are to be engaged by the weapon platforms. This field can be entered by the user.

Percent Tracked. The percentage of the targets tracked, as specified in the sensor display for the selected range bin.

Tracked. The number of each threat type tracked in the selected range bin. The number is the integer value of the product of the number times the percentage tracked.

Percent Engageable. The percentage of targets that can be engaged by the weapons. This field, which can be entered by the user, is intended to account for targets that may be out of weapon range, or placed by other factors outside the weapon's capability to engage.

Engageable. The number of targets engageable by the weapons. The number is the integer value of the product of the number tracked times the percentage engageable.

Selection of a line allows the user to modify the priority and percentage engaged fields.

• **Weapon Platform Box.** This box permits the review of the types of weapon platforms in the selected range bin, and the specification of the percentage of the weapons available to engage targets, and the order in which the weapons will be used. The fields are described below.

Weapon Platform. The types of weapon platforms for the selected range bin, as specified in the weapons platform display.

Total. The total number of available weapon platforms as specified in the weapon platform display for the selected range bin.

Percent in Battle. The percentage of the weapon platforms that are available to engage the targets. The field is user-enterable. The percentage is intended to represent the percentage of the total force that has a battle-ready status.

Number. The number of weapon platforms in the battle. The number is the integer value of the product of the total times the percentage in battle.

Priority. The order in which the weapons are to be utilized. This field can be entered by the user.

Selection of a line allows the user to modify the percentage in battle and priority fields. It also places the selected weapon platform's armament data in the armament box.

• **Armament Box.** This box supports the review and entry of the order in which the specific weapons on the weapon platforms are used. The fields in the armament display are described below.

Armament Type. The types of armament carried on the selected weapon platform. The data in this field are based on data from the weapon platform display.

Number. The number of each respective armament type on the weapon platform. The data in this field are based on data from the weapon platform display.

Priority. Allows the user to specify the order in which each type of armament will be expended by the platform.

Selection of a line allows the user to modify the priority field.

- *Target Summary Box.* This box contains the results of the SWaT calculations for targets in a selected range bin. The fields are described below.

Target. The target types involved in the SWaT calculations.

Killed. The number of targets of each type calculated by the SWaT model to have been killed.

Remaining. The number of targets of each type calculated by the SWaT model to be remaining, including targets that were engaged as well as those that were not engaged or tracked.

- *Armament Summary Box.* This box contains the results of the SWaT calculations for the armament in a selected range bin. The fields are described below.

Armament. The armament types involved in the SWaT calculation.

Used. The number of shots of the armament type fired.

Remaining. The number of each type of armament remaining.

- *SWaT Evaluation Usage Instructions.* The user selects the SWaT evaluation from the main menu bar and is presented with the SWaT display form initialized at range bin 1. It should be noted that prior to entering the SWaT form the user should already have entered range bins, probability of tracking, weapon types, armaments, probability of kill, threat platform data, and threat deployment range information with the sensors, weapons, and threat forms.

A typical sequence of steps for utilizing the SWaT form is as follows.

1. User starts at range bin 1 (initial value), selects a TACTIC (single shot or shoot-look-shoot), and TRIAL 1 (initial value).

2. In the target box, select each of the threats and enter the desired engagement priority and percent engageable.

3. In the weapon platform box, select a platform and notice that the armaments box is initialized with the appropriate armament for the selected platform.

4. Enter the percentage in battle and the priority for the selected weapon platform.

5. Move to the armament box and set the priority for each of the armaments.

6. Repeat steps 3, 4, and 5 for all weapon platforms in the weapon platform box.

7. Move to the application buttons box and select CALC; observe the target summary and armament summary boxes for SWaT calculation results.

8. To recalculate the SWaT results with different priorities or percentage, select NEW TRIAL and repeat steps 2 through 7.

9. Upon completion of the desired number of trials, click on the TRIAL indicator until the desired results appear in the form. Select STORE to save this TRIAL for input to the subsequent range bin. All other trials will be discarded. This completes the analysis for one range bin.

10. To proceed to a subsequent range bin, select the range bin indicator to move to the next range bin.

11. Repeat steps 2 through 8 for each subsequent range bin.

B4.2.10 GOAL EVALUATOR FORM. The goal evaluator form (Figure B-13) allows the user to subjectively measure the capabilities and attributes of a C^2 system against the set of goals developed for it with the goal developer form.

- *Standard Applications Box.* Clicking the mouse on the following buttons in the standard applications box has the specified results.

```
GOAL EVALUATOR
┌─────────┐ ┌─────────┐ ┌─────────┐
│  HELP   │ │  DONE   │ │ CANCEL  │
└─────────┘ └─────────┘ └─────────┘
GOAL GROUP        GENERAL GOAL              EVALUATION        ┌─────────┐
─────────────     ────────────────────     ──────────        │ MODIFY  │
System Robustness Maximum Case Attack       Unsatisfactory    ┌─────────┐
System Robustness Processing for all Regions Satisfactory     │ DETAILS │
System Robustness Conserve Defensive Resources Unsatisfactory ┌─────────┐
System Robustness Expend Resources Proportionall Partly Satis │  STORE  │
Surveill Data Use Use all Data Received      Partly Satis     ┌─────────┐
Surveill Data Use Not Rely on Deniable Data  Unsatisfactory   │ REVERT  │
                                                              └─────────┘
```

Figure B-13. Goal Evaluator Form

HELP. Invokes the help system if it is not already invoked. Displays the text of the relevant section of this User Manual as the user moves the cursor into a box (except, for technical reasons, a box in which a textual description or textual remarks are displayed). A second click on the button turns this capability off; meanwhile the button remains inverted.

DONE. Removes the form from the screen, and, in developer mode, reports STOREd changes to the knowledge server. Changes that are not STOREd are lost.

CANCEL. Removes the form from the screen. Even STOREd changes are lost.

• *Goal Group Box.* This box allows the user to select the goal group of interest. Clicking the mouse on the goal group results in the short names of the goals in the selected group being displayed in the goal name box.

• *Goal Evaluation Box.* This box permits the user to review the evaluation of each goal. The fields are described below.

Goal Group. Shows the group in which the general goal falls.

General Goal. Shows the name of a general goal.

Evaluation. Shows the evaluation of the goal.

• *Goal Evaluation Application Buttons Box.* The application buttons allow the user to modify the goal status as a result of changes made to the baseline and to save the resultant changes. The buttons are described below.

MODIFY. Allows the modification of the evaluation data for the selected general goal.

DETAILS. Displays the goal developer form initialized to the selected general goal, or initializes the goal developer form to it if it is already on screen.

STORE. Saves the changes made during interaction with the form.

REVERT. Discards any changes made since the last time the STORE button was pressed, or the opening of the form if it has not yet been pressed.

APPENDIX C
SELECTION OF THE LOGIC PROGRAMMING LANGUAGE

To select the logic programming language for the CMLP project we developed evaluation criteria, identified available Prologs, performed a preselection against the three most important criteria (commercial availability, robustness, runability on VAX/VMS as well as Sun 3/Unix). and from the languages remaining selected one by evaluation against the full set of criteria.

Eleven logic programming languages were on our initial list:

- ALS
- BIM
- CLP(R)
- EqL
- Horne
- IF
- MProlog
- PARLOG
- Quintus
- Rhet
- Trilogy

We selected five for closer examination: ALS, BIM. IF, MProlog, and Quintus. We eliminated ALS Prolog, an exciting new research direction for logic programming, because at the start of the project it had not completed the transition from research vehicle to robust commercial product. Preliminary review narrowed the list down to Quintus and BIM Prologs for in-depth evaluation. The detailed results of their evaluations against the criteria presented in Section C1 are given in Sections C2 and C3, respectively. Quintus and BIM were both strong products. We selected Quintus because we had extensive previous experience with it, hence firsthand knowledge of its quality and robustness; felt that the Quintus library gave it a significant advantage; and had reservations about the level and quality of support available for the BIM product.

C1. EVALUATION CRITERIA

Twelve issues were postulated as the evaluation criteria for the CMLP logic programming language.

1. Documentation.

a. Tutorial. Is the introductory manual suitable for non-Prolog people?

b. User's Guide. Though higher level than tutorial, is it still easy to use?

c. Reference Guide.

2. Efficiency. General performance considerations.

a. Hashing/Indexing, Etc. Is there some scheme for speeding access to clauses? How many different arguments can be indexed? How much space does an index take?

b. Speed. How fast? How many LIPS?

c. Memory Management. Does it do any? Is it dynamic (grow on demand) or static (have to rebuild)? Does it garbage-collect clause space? The heap? The other stacks? Can it use virtual memory?

d. Compilation Strategy. Can a compilation be saved on disk? Is the compiler a separate program?

3. Robustness/Bugs. Does it have bugs? Does it dump core?

4. Vendor Support. How good is it?

a. Corporate Resources. How many people work on it? How big is the company? How sound is the company? How long has the company been in business? What is the nature of current development efforts (what improvements are coming)?

b. Location. Vendor should either be located in the U.S. or have a sound support network; otherwise getting timely support will be too big a problem.

5. Development Environment. Is the environment a separate product?

a. Editor Interface. Is it built in, or a separate product?

b. Lint. Is there a lint program? Is there a cross-referencer?

c. Compiler. Is there a separate compiler?

d. External (C Routines) Linking. Does it allow linking external (foreign) routines? Is it part of a separate linking process. How easy is it?

e. Deliverable Executables. Can deliverable executables be built? What are the licensing considerations?

6. **Portability.** How portable is it? Does it run on a Sun/3? Sun/4? VAX? PC? Any other machine?

7. **Graphics.** What kind of graphics does it support? What graphics environment does it run in?

a. Windowing. Does it run in multiple windows? Which windowing system: X? NeWS?

b. Popup/Pulldown Menus/Icons. Can it get a menu? What kinds of menus?

c. Editing Within Windows. Is editing integrated?

d. Mouse Support.

e. Function Keys.

f. Library Support. Are graphics in a library? Can some vendor-supplied library be linked in?

8. **Database.** What is the nature of any database it may have?

a. Internal/Memory Based. Does it support Retract and Erase? Does it provide additional database-like features?

b. External/Disk Based. Is there an external database interface? Can any database library (e.g., UNIFY) be linked in easily? Are there any extensions to Prolog that use databases well?

9. **Modularity.** Support for engineered and reusable code.

a. Modules. Does it have modules? Do the modules provide for interface definitions?

b. Modes. Does it have modes? Does it use modes to speed compiled code? Does it use types semantically?

10. **Debugging Environment.** What is the nature and extent of the debugging environment?

11. **Conformance to Standards.** What standards does it come closest to now? What standard will it choose?

a. Edinburgh DEC/10. The quasi-standard shared among several Prologs.

b. Prolog II, European. Used by Prolog II.

c. BSI. Standard being developed by the British Standards Institute. Has been adopted by ISO. Supposed to be derived from Edinburgh. R. A. O'Keefe has had many problems with it.

12. **Predicate Library.** Does it provide a library of Prolog predicates?

C2. BIM EVALUATION

1. **Documentation.** The documentation is reasonable, but uses an illogical layout.

a. Tutorial. None.

b. User's Guide. None.

c. Reference Guide. Split into sections on basics, external language, debugger, user interface, and database.

2. **Efficiency.**

a. Hashing/Indexing, Etc. Can index on any three arguments in compiled code. Obviously takes more space the more indexes there are.

b. Speed. Compiled to native code (i.e., MC68020 or VAX). We have not run benchmarks, but BIM claims to be faster than Quintus.

c. Memory Management. Fixed allocations at startup time, so if you run out you must abort and start again. Garbage collection: BIM collects clause space and stacks. Virtual memory.

d. Compilation Strategy. BIM compiles to native code and has intermediate files ".wic" to hold the results of compilation. There is a separate compiler program.

3. **Robustness/Bugs.** We encountered no serious bugs or core dumps.

4. Vendor Support.

a. Corporate Resources BIM is a small company with split priorities. It has fewer than eight people working on Prolog development. Soundness is unknown. Current development efforts include adding the record family of predicates, compiler polishing, support for conversion from other Prologs, improved SunView-based debugger.

b. Location. In Belgium; support via San Diego. The main expertise of the company is clearly in Belgium, so timeliness and expertise of support are potential problems.

5. Development Environment. No known advantages/disadvantages as compared to Quintus. BIM interfaces C routines from inside Prolog execution at run time and has no deliverable executables without special arrangements.

6. Portability. Runs on Sun (main effort) and VAX, not on PC or other systems.

7. Graphics. BIM's entire approach to graphics is to give access to the low-level libraries. In particular, it already has the SunView interface worked out. BIM works well with SunView but gives no additional support. BIM has no plans for XWindows support, but we could built the library interface. SunView supports all subcriteria (windowing, popup/pulldown menus/icons, editing within windows, mouse support, function keys, library support).

8. Database.

a. Internal. No information available.

b. External. An interface to the Unify database library.

9. Modularity.

a. Modules. BIM has modules for grouping code, not implementation hiding. There is no interface definition.

b. Modes. BIM has modes; also uses them to specify indexing.

c. Types. None.

10. Debugging Environment. BIM has an elaborate SunView-based debugger. It mimics Sun's dbxtool to some degree and will do a better job once BIM gets it running as two processes.

11. Conformance to Standards.

a. Edinburgh DEC/10. Not at all. There is a compatibility switch that supposedly makes it takes C-Prolog syntax.

b. Prolog II, European. Normal style closer to this.

c. BSI. BIM is part of this European group which is trying to standardize Prolog.

12. Predicate Library. None.

C3. QUINTUS EVALUATION

1. Documentation. Very high quality. Has been the standard against which we judge other documents. It breaks up the documentation somewhat differently from tutorial/ user's guide/reference, but not appreciably.

a. Tutorial. No separate tutorial as such, but the first part of the user's guide is a tutorial.

b. User's Guide. A very comprehensive user's guide. In addition, the library manual and windows manual contain some user's guide material.

c. Reference Guide. The reference manual is quite good. There are separate manual sections for system-dependent features, the predicate library, windows, and the database.

2. Efficiency. General performance considerations were as follows.

a. Hashing/Indexing, Etc. Quintus builds a hash index on the first argument of a clause. It also documents how more indexes can be used by adding secondary predicates.

b. Speed. Seems fast enough, but we did not perform exhaustive benchmarks.

c. Memory Management. Has excellent memory management that allocates additional space as needed. Garbage-collects clause space and the stacks. Uses virtual memory.

d. Compilation Strategy. Compilations cannot be saved, but whole executables can. The compiler is built into the normal executable.

3. *Robustness/Bugs.* No known bugs or core dumps from Prolog. UniPress emacs, which is included, will dump core on occasion. UniPress emacs also has some bugs and infelicities that are irritating.

4. *Vendor Support.* Phone support 8 hours a day.

a. Corporate Resources. More development and support personnel than BIM. Quintus seems to be a well-financed company and has been in business for quite some time. Current development efforts are constantly improving the product.

b. Location. Mountain View, California.

5. *Development Environment.* Development environment is a combination of Prolog and UniPress emacs. Its problems stem more from UniPress than from Quintus.

a. Editor Interface. Tight; Prolog runs in an editor window.

b. Lint. There are lint and cross-reference utilities.

c. Compiler. Compiler is part of the Prolog executable (compile/1).

d. External (C Routines) Linking. Allows for linking foreign routines that follow the C, Pascal, or Fortran calling conventions. Linking is done during program execution and runs off a simple set of Prolog predicates.

e. Deliverable Executables. Deliverable executables can be built using the PAC (Prolog Application Compiler). License is needed for each deliverable sold.

6. *Portability.* Runs on many machines; Quintus presumably wants it to run on even more.

a. Sun. Runs on Sun/2, Sun/3, and Sun/4.

b. VAX. Runs on VAX/VMS and VAX/Ultrix.

c. PC. Runs on IBM RT PC running AIX.

d. Others. Xerox 1100 (Lisp), Apollo, NCR Tower, IBM 370 (MVS and VM/CMS).

7. *Graphics.* Quintus runs a separate package called ProWINDOWS.

a. Windowing. ProWINDOWS does build windows. They are built on top of SunView, but Quintus plans to implement this on top of X for greater portability.

b. Popup/Pulldown Menus/Icons. Menus, icons, texts, etc., are all available.

c. Editing Within Windows. Editing of SunView text subwindows may be supported.

d. Mouse Support. Yes.

e. Function Keys. Yes.

f. Library Support. Could link X. NeWS, or SunView libraries, but Quintus does not give up control of program execution easily, so SunView has problems when trying to run in one process.

8. *Database.*

a. Internal/Memory Based. Quintus has the record and erase predicates, as well as clause/3 and assert/3 with references.

b. External/Disk Based. Tight interface to SunUnify that makes any external relations look like internal predicates. A limited form of access to the database power is provided by compiled views, but the interface definition appears to put quite a burden on the Prolog user of the database.

9. *Modularity.* There is very good modularity support.

a. Modules. Quintus has modules that provide for interface definitions.

b. Modes. Quintus has mode declarations, which are ignored.

c. Types. There is a type checker, written by O'Keefe, in the public domain.

10. *Debugging Environment.* Fairly nice implementation of the box model tracer with spy points.

11. *Conformance to Standards.* Quintus is itself a de facto standard, closely matching DEC/10, Edinburgh. Quintus will probably support other standards as they emerge.

a. Edinburgh DEC/10. This is the standard closest to Quintus.

b. Prolog II, European. Does not conform.

c. BSI. R. A. O'Keefe has many problems with this standard; hence Quintus is unlikely to do much with it.