DESIGN INSTITUTE REPORT: CADRU-11-97
3rd Printing, April 2003

Decision-Support Systems: Notions, Prototypes, and In-Use Applications

With Emphasis on Military Applications

Jens Pohl
Art Chapman
Kym Jason Pohl
Jonathan Primrose
Adam Wozniak

Collaborative Agent Design Research Center (CADRC)
California Polytechnic State University, San Luis Obispo, CA 93407

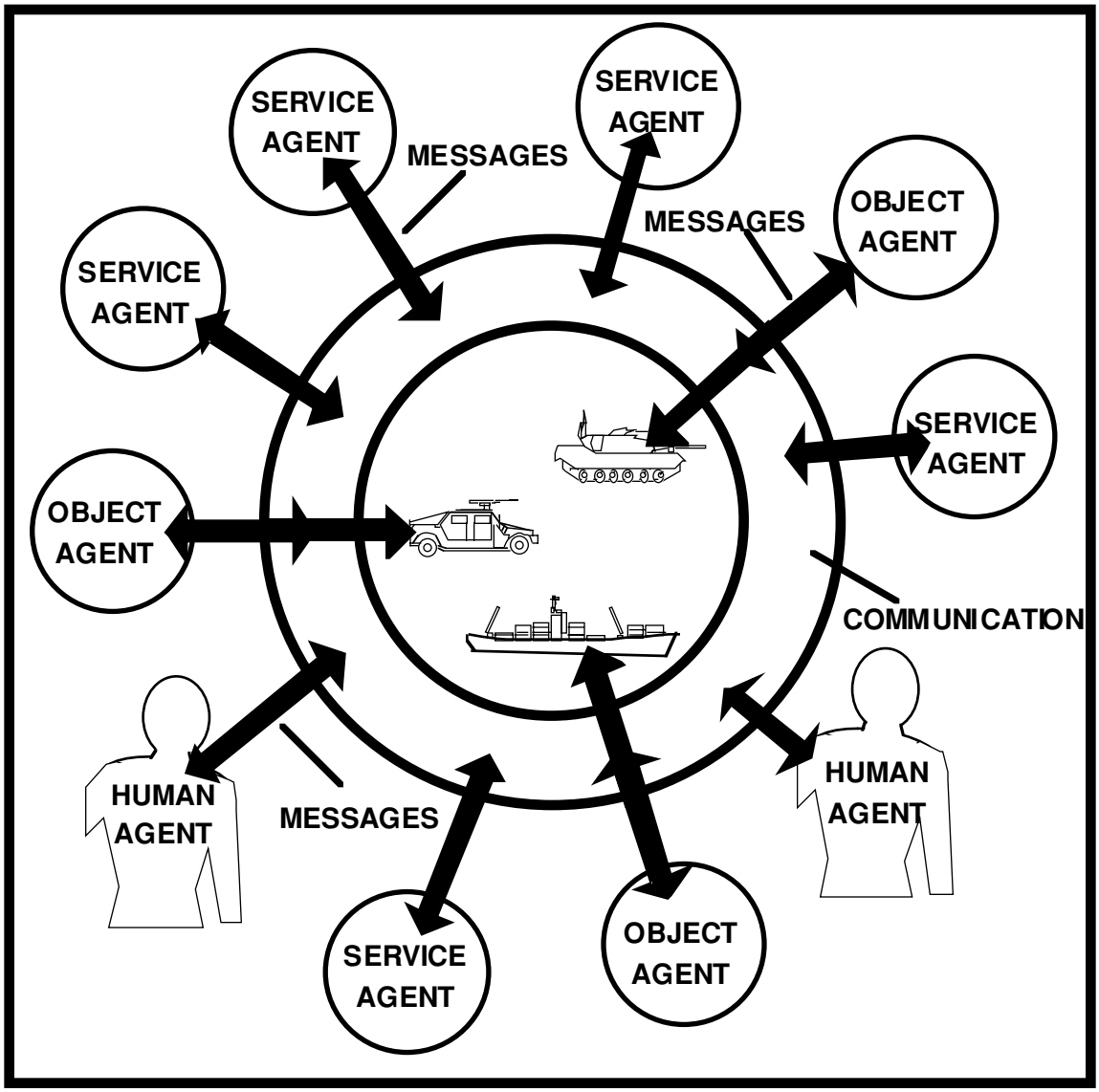
Abstract

This report describes work performed by the Collaborative Agent Design Research Center (CADRC) over the past several years in the design and implementation of collaborative, computer-based, decision-support systems, mostly for military applications. In these systems multiple components, either program modules or separate processes (i.e., software agents), cooperate with each other and human decision makers to solve complex problems. The components are essentially of two types: knowledge-based narrow domain experts that provide services to other agents (i.e., service-agents); and, more autonomous agents (i.e., object-agents) that represent the interests of selected objects in high level information representation schemas.

Based on the notion that all computer programs are essentially agents, the report traces the evolution of 1st and 2nd Wave software from single agent, stand-alone decision-support applications to integrated, collaborative, distributed, multi-agent decision-support systems. Several multi-agent decision-support systems developed by the CADRC over past years are described. These include: ICODES (Integrated Computerized Deployment System) for ship load planning; CIAT (Collaborative Infrastructure Assessment Tool) for facilities management; FEAT (Force Employment Analysis Tool) for military planning and engagement coordination, and KOALA (Knowledge-Oriented Object-Agent Collaboration) for architectural space planning.

Keywords

agents, blackboard, buildings, cargo, collaboration, C2, C4, C4I, CIAT, command and control, communication, coordination, data-blackboard, decisions, decision-support, design, distributed, facilities management, FEAT, ICODES, interactive, knowledge, KOALA, load-planning, message-agents, military, monitor agent, objects, object-agents, planning, representation, service-agents, space planning, soldier, ships, stow-planning, transportation, warfighting



Distributed, Integrated Multi-Agent Decision-Support Systems

Decision-Support Systems: Notions, Prototypes, and In-Use Applications

With Emphasis on Military Applications

1. Decision Systems: Background and Introduction

Decision-making is a problem solving activity that human beings undertake on a daily basis in all of their endeavors. Although there are many definitions of decision making, depending on the goals, beliefs, and current knowledge of the researcher (Frensch and Funke 1995), it is generally agreed that decision making is a goal-directed activity that involves a wide range of cognitive operations and that the specific process and strategies employed by individual decision makers can vary widely.

The work of the Collaborative Agent Design Research Center (CADRC) described in this Technical Report is specifically focused on 'complex problems' and computer-based decision systems that are designed to assist, not replace, human decision makers in the solution of these problems. We consider the relative level of complexity of a problem to be a primary function of the number and strengths of the inter and intra relationships that exist among internal and external components of the problem, and the degree of uncertainty that surrounds the definition of these components. Typically, complex problems involve many strong relationships among internal components as well as important dependencies on external factors. The external factors may be determined by events, the cause of which could be unrelated to the problem situation. For example, in planning a counter offensive a military commander may have to consider not only the many variables and their interrelationships that impact actual battlefield conditions (e.g., enemy and friendly forces positions and capabilities, battlefield terrain, weapon capabilities and availability, weather, etc.) but also higher level political considerations on both sides that may influence enemy actions and/or the ability of the commander to execute his or her own strategies.

A more specific example of the dependency on external factors is a fairly common occurrence in the logistical transportation field. Cargo specialists may spend up to two man-days to design a cargo load plan for a ship, a complex undertaking that involves many interrelationships among issues ranging from the trim and stability constraints of the vessel, hazardous material segregation requirements, lift capabilities, to loading sequences and stow area accessibility restrictions. External dependencies include the availability of port facilities (e.g., mobile port cranes, electrical lighting for nighttime loading operations, etc.), port traffic conditions that may impact the movement of cargo from staging areas to the pier, labor relations that will influence loading operations, and the arrival condition of the vessel to be loaded. The latter can vary significantly from the expected. Such variations may range from the inoperability of specific ship equipment (e.g., onboard cranes) to the amount and actual location of pre-loaded cargo. It is even possible that the vessel that arrives at the port is not the vessel that was considered during the planning stage. The factors that may have forced a change in vessels are likely to be quite independent of the internal problem conditions. For example, the original ship may have broken down in transit, or it may have been required for other purposes that took precedence for reasons unrelated to either the destination of the cargo or the purpose of the planned loading operation.

As shown in this example, uncertainty in complex problems extends beyond the lack of definition of the individual problem elements, such as hazardous material considerations and stow area accessibility, to the relationships of these elements to each other and external factors (e.g., replacement of the expected vessel with another vessel). In other words, the dynamic information changes that are characteristic of complex problems tend to modify, delete and create new relationships among both the internal elements and the external dependencies of the problem situation. Even a relatively small change in one element can trigger a series of major relationship changes that may essentially restructure the entire problem. This interconnectedness of complex problem situations poses particular difficulties to the human cognitive system, because it forces the decision maker from the normal sequential paradigm into a parallel reasoning process.

Heightened expectations of quality, accuracy, execution speed, and responsiveness to dynamically changing conditions are increasingly challenging the capabilities of human decision makers in the many complex problem situations that they face in their varied endeavors. It is therefore not surprising that mankind should be increasingly looking to technology in the form of computer-based decision-support systems, for assistance. Such assistance would appear to be appropriate and welcome in at least the following functional areas:

1. To provide access to factual data that describe past and present conditions of dynamically changing aspects of the problem situation (e.g., changes in enemy positions, weather conditions, resource consumption, etc.).

2. To provide access to relatively static reference information (e.g., cost rates, equipment characteristics, etc.).

3. To provide access to existing knowledge and specialized expertise in domains that are relevant to the problem situation. This knowledge may range from standard practices and procedures (i.e., prototype knowledge-bases (Gero et al.1988, Rosenman and Gero 1993, Pohl and Myers 1994, Pohl et al. 1988)) to rule-based sequences and strategies that are commonly applied by human experts to similar problem conditions.

4. To assist in the analysis and fusion of information derived from multiple sources for purposes of establishing and maintaining an accurate view of the current state of the problem (i.e., 'situation awareness' in the military environment).

5. To alert the human decision maker to possible conflicts and transgressions of boundaries (i.e., violations), based on parameters that may be modified from time to time.

6. To propose alternative solution strategies and identify opportunities for pursuing specific directions.

7. To provide explanations of how and why particular recommendations and conclusions were generated by the components of the decision-support system.

8. To learn from the interactions between the human decision maker and the decision-support system the methods and strategies that the former employs in particular problem situations, and to be able to apply these methods and strategies in the absence or on the instructions of the human decision maker.

The human decision maker brings a complex interplay of many cognitive, motivational, personal, and social factors into the human-computer partnership. Most of these factors are poorly understood, being based on neuro-physiological, biological and behavioral processes that are still largely undeciphered. This requires a great deal of flexibility to be built into the user-interface so that the partnership can evolve in directions and capabilities that cannot be predetermined at the outset.

1.1 The Importance of the Environment

One might wonder what an individual who has enjoyed the highest level of education at the conclusion of the 20th Century could achieve, if he or she were to be transplanted to some distant past such as the late Middle Ages (11th Century) or the early Renaissance (14th Century). After some thought one would probably arrive at the surprising conclusion that the potential for making a major contribution would be limited at best. The ability to perform at a high intellectual and productive level, particularly in technical areas where this individual may function most effectively today, is largely dependent on the availability of a supportive infrastructure. In other words, to apply existing knowledge and develop new knowledge in a particular field of expertise requires an environment that provides appropriate tools, materials, information resources, and also imparts a perceptible sense that such contributions are needed and welcome.

The problems of this transplanted individual would stem not so much from the absence of an industry that can produce the tools, such as computers, that the individual may have come to rely on for most of his or her daily endeavors, but more importantly from the inadequacies of the available human interactions. In our society today complex problems are typically solved, not by a single individual, but by the cooperation of several persons with compatible objectives. In fact, much of the technological development in recent times has been focused on optimizing the accessibility of human and other resources to the individual.

In considering the time scale of evolution it can be argued that problem-solving success, particularly in respect to the acquisition and application of expert knowledge, is greatly accelerated by a supportive environment. It took over three billion years for homo sapiens to evolve after single cell entities emerged from the primordial soup, some two million years for the emergence of agriculture, only another 10,000 years for the invention of writing, less then a few hundred years for industrialization (Dawkins 1987, Brooks 1990), and, a mere 40 years for computerization to affect virtually every aspect of our

lives. It appears that our ability to interact with a dynamic environment allows us to collectively achieve advances that are beyond the capabilities of the individual.

Equally supportive of these arguments is the notion that information is not stored in the human brain in specific neuron templates, but rather 'evoked' through a state of the entire nervous system (Wittgenstein 1953, Dreyfus and Dreyfus 1988, Maturana et al. 1960, Winograd and Flores 1987). Analogously, if we consider the individual as an agent cooperating within an environment of many agents, it would appear reasonable that the collective intelligence of the environment exceeds that of the individual. This hypothesis is indeed supported by recent work dealing with the dynamic behavior of computer-based agents interacting in a computational ecology (Huberman and Hogg 1988). It has been shown that in such systems the cooperative interactions among agents operating in different task domains can lead to the improved performance of the system as a whole. Such gains are measurable, for example, in situations where multiple agents conduct database searches in parallel for information items that can satisfy certain constraints. The overall search time is determined by the agent that finds an acceptable answer first, thereby terminating the concurrent searches of the other agents.

Huberman (1991) draws attention to the fact that this characteristic of large cooperative systems to achieve a higher level of performance than could be predicted on the basis of a detailed analysis of their component tasks has been observed in a fairly wide range of applications (Aitchison and Brown 1957): such as research productivity (Schockley 1957); economics (Montroll and Shlesinger 1982); and, ecological diversity (Krebs 1972).

1.2 The Influence of Intuition and Emotions

The ability to analyze problem situations, reason about solution strategies, and develop one or several alternative courses of action is a fundamental human cognitive skill. This skill has and will continue to evolve as human beings interact with their environment and challenge themselves to understand, predict and control phenomena and events of increasing complexity.

In this environment complexity is a function of the many interrelationships that influence the nature and behavior of the factors that we identify as being pertinent to a given situation. In fact, the process of making decisions is mostly concerned with unraveling these interrelationships, a task that is pervaded by difficulties. First, there is a need for establishing some solution objectives to provide a direction for determining priorities and an orderly sequence of actions. However, the ability to establish objectives presupposes at least some level of understanding of the problem situation. In other words, at least the vestige of a conceptual solution, even if only in terms of an intuitive feeling about the kind of solution that is likely to eventuate, will be formed by the decision maker during the earliest stages of the solution process. The existence of this conceptual solution is both advantageous and disadvantageous.

An early conceptual solution is helpful and arguably an essential prerequisite for defining the framework within which explorations of the problem situation and the decision

making process at large, will proceed. Without such a framework, in the realm of spontaneous, unsystematic explorations of aspects of the problem, the human cognitive system tends to perform unevenly and unpredictably at best.

While there is much historical evidence that the early formulation of a conceptual solution can be the decisive factor in the realization of a timely final solution, there are also outstanding examples to the contrary. Early commitment to a solution path can introduce biases and misconceptions that will lead to contrived solutions that become weaker and weaker as more and more information about the problem situation becomes available. The decision makers are faced with a dilemma: discard the original concept, or modify an increasingly flawed concept to bring it into closer alignment with the perceived situation. Political and emotional factors from both outside and within the problem solving team will inevitably emerge to fuel the dilemma. A well known example of such a problem situation was the insistence of astronomers from the 2nd to the 15th Century, despite mounting evidence to the contrary, that the heavenly bodies revolve in perfect circular paths around the earth (Taylor 1949). This forced the astronomers to progressively modify an increasingly complex geometric mathematical model of concentric circles revolving at different speeds and on different axes to reproduce the apparently erratic movement of the planets when viewed from Earth. Neither the current scientific paradigm nor the religious dogma of the church interwoven within the social environment allowed the increasingly flawed conceptual solution of Ptolemaic epicycles to be discarded. Despite the obviously extreme nature of this historical example, it is worthy of mention because it clearly demonstrates how vulnerable the rational side of the human cognitive system is to emotional influences.

This does not mean that it would be best to strive to remove the human element altogether from decision making systems. On the contrary, particularly in complex problem situations where there tends to be a significant element of uncertainty, human intuition and emotions are not only desirable but often necessary ingredients of a successful outcome. In any case, for valid reasons, human beings are unlikely to trust themselves completely to the decisions made by machines for many years to come, if ever.

A second difficulty that faces problem solvers as they attempt to identify interrelationships is their inability to fully define the problem. The problem situation is likely to include factors that are unknown at the time when a solution is desired. This means that parts of the problem are not understood and in particular, that the relationships among these parts and the known parts of the system cannot be explained. Still worse, these unknown factors will influence other apparently 'known' relationships with misleading results. In other words, the decision makers may believe that they understand certain relationships but are in fact misled by the influence on these relationships of other unknown factors. One can argue that it is an intrinsic characteristic of complex problems that they are never fully defined, nor are they ever fully solved, because they constantly mutate as the issues and forces that feed them change.

1.3 The Role of Leadership

Historically, in the field of management, decision-making has been exercised within a framework of hierarchical authority. It was held, and this continues to be a somewhat fundamental notion in corporate, government and military organizations, that important decisions can be made only by persons who have the authority to make such decisions. This authority is typically vested in position, rank, and ownership, on the a priori assumption that knowledge and problem solving abilities are demonstrated prerequisites of persons attaining such stature.

On closer examination this would appear to be a rather simplistic and limiting view of the real world. This notion of decision-making places an emphasis on process with the objective of exercising control over both the contribution of the participants and the tempo of the problem solving activities. It implies a deep-seated fear that errors in judgment introduced at the lower levels of the hierarchy can easily and decisively mislead the general direction of the solution path. It further suggests that the decision-making process itself should be hierarchical in nature. Neither of these contentions would appear to be valid. First, due to the continuous information changes that are characteristic of complex problems, there is a need to maintain a high level of responsiveness and openness. While the information changes may enter the system from any direction, they are more likely to be detected at the operational levels first and then percolate through to the management levels. However, management has a tendency to suppress these changes when they negate or interfere with the current view of the situation or run counter to a predetermined course of action.

Second, the hierarchical structure itself seriously constrains the initiative and contribution potential of the lower levels. Yet these operational levels are normally closest to the source of the information changes that drive the decision-making process and are therefore in a good position to interpret and judge the relevancy of their observations. Third, a hierarchical decision-making process is by its very nature designed to control the vertical flow of information. The information channels are typically laid out in pipeline fashion on the assumption that the information flow will be progressively filtered and reduced in volume toward the upper echelons of the pyramid. This is necessary to avoid communication bottlenecks at the highest level where the decisions will be made. Unfortunately, in practice, the opposite usually occurs. For example, during military operations commanders tend to be overwhelmed by the shear volume of information that competes for their attention. The lower levels, being mainly authorized to collect and pass on information rather than analyze and interpret what they collect, will be reluctant to exercise initiative in case their actions will contravene the chain of command.

In this environment information is viewed as a commodity that is 'owned', to be made available on a limited basis typically only when the owner is directed to do so. Under these circumstances information tends to flow: upward, mostly on request and when the owners feel that their objectives will be served without jeopardizing their status and position in the hierarchy; downward, based almost exclusively on directions and authorizations received from above, mostly in support of execution orders; and laterally,

within a network of domain specific activities that is often governed predominantly by informal relationships. Clearly in this model the information flow is severely restricted by the organizational structure. The hierarchical model places paramount importance on organizational leadership, on the assumption that the problem exists mainly for the organization and that the problem solving objectives are therefore subservient to the objectives of the organization. In fact, this assumption is difficult to defend. Usually organizations, whether commercial, government or military, exist for the purpose of serving and/or protecting the welfare and interests of others. It therefore follows that the objectives of the organization should be subservient and adaptable to the needs of the problem situation. The structural notions of organizational leadership and information ownership are relevant to the problem situation only to the extent that they facilitate the solution of the problem.

More relevant to decision making in complex problem situations is the notion of situational leadership. The need for this kind of leadership arises whenever any of the participants in a problem-solving task see an opportunity for actions that will accelerate the completion of their own tasks and/or contribute to the tasks of others. In this respect situational leadership assumes a non-hierarchical cooperative operational structure in which the participants collaborate freely within the existing organizational levels. Under these circumstances the purpose of organizational leadership is to support and not to dictate the problem solving process; to remove obstacles and empower the individual problem solvers, rather than control their participation and the tempo of their contributions. In particular, the role of the organizational leadership is to prevent anarchy by guiding the situational leaders toward consensus. Naturally each situational leader cannot be the sole judge as to his or her contributions to the tasks of others. However, situational leadership is akin to initiative and should be encouraged to occur at any node of the problem system regardless of the organizational position or level of the person exercising the initiative. It is a spontaneous response to the current state of the problem, as viewed from a particular node that maximizes concurrent problem solving activities.

Problem solving is a collaborative activity that dynamically develops its own supportive structure in direct response to the current needs, restrictions and opportunities of the problem system. To constrain this decision-making activity within the rigid framework of an hierarchical organizational structure inhibits those human capabilities, such as exploration, experimentation, initiative and intuition, that have been found to be among the most effective problem solving skills. Typically, the evolving structure assumes a flattened network configuration with both nodes and inter-node communication channels appearing and disappearing spontaneously, driven almost entirely by the changing context of the problem situation. In this network the relative strengths of relationships and the relative importance of nodes changes readily in response to factors that are largely independent of any predetermined organizational leadership structure. Schmitt (1997), in discussing maneuver warfare, presents strong arguments in favor of asynchronous military operations in which the various components of an operation are not synchronized to occur in a predetermined order (i.e., in unison). He presents the example of a soccer team, "... 11 players, each with assigned responsibilities but acting independently..." as the situation on the field offers and demands. While there are preset

plays and team strategies "...the players react individually to the ball, and yet somehow the result is that they manage to work together as a team".

1.4 Guiding Principles for the Design of Decision-Support Systems

Based on these introductory comments and our experience with the design and implementation of decision-support systems over the past decade, we have identified the following general guiding principles. These evolving principles have and will continue to serve as a framework for most of the work of the CADRC and are therefore reflected to some degree in all of the systems described in this report.

1.4.1 Emphasis on Partnership

A successful decision-support system is one that assists rather than replaces the human decision maker. Human beings and computers are complementary in many respects. The strengths of human decision makers in the areas of conceptualization, intuition and creativity are the weaknesses of the computer. Conversely, the strengths of the computer in computation speed, parallelism, accuracy and the persistent storage of almost unlimited detailed information are human weaknesses. It therefore makes a great deal of sense to view a decision-support system as a partnership between human and computer-based resources and capabilities. Automation should be restricted to the monitoring of problem solving activities, the detection of conflicts, and the execution of evaluation, search and planning sequences.

In this partnership a high level of interaction between the user and the computer is for several reasons a necessary feature of the decision-support environment. First, it allows the user to assist the computer in the recognition and interpretation of the real world object representation that forms a basis of any meaningful communication between the user and the computer. Second, it provides opportunities for the user to guide the computer in those areas of the decision-making process, such as conceptualization and intuition, where the skills of the user are likely to be far superior to those of the computer. Particularly prominent among these areas are conflict resolution and risk assessment. Third, it establishes a readily available channel through which the user can enter new and modify existing information as desired to maintain a high level of spontaneity in the partnership.

1.4.2 Collaborative and Distributed

Complex problem environments normally involve many parties that collaborate from widely distributed geographical locations and utilize information resources that are equally dispersed. The decision-support system can take advantage of the distributed participation by itself assuming a distributed architecture. Such an architecture typically consists of several components that can execute on more than one computer. Both the information flow among these components and the computing power required to support

the system as a whole can be decentralized. This greatly reduces the potential for communication bottlenecks and increases the computation speed through parallelism.

Another advantage of the distributed approach is the ability to modify some components of the system while the system as a whole continues to operate with the remaining components. Similarly, the malfunction or complete failure of one component does not necessarily jeopardize the entire system. This is not so much a matter of redundancy, although the distributed architecture lends itself to the provision of a high degree of redundancy, but rather a direct result of the physical independence of the components. While the components may be closely integrated from a logical point of view they can operate in their own autonomous physical environment.

1.4.3 An Open Architecture

The high degree of uncertainty that pervades complex problem environments extends beyond the decision-making activity of the collaborating problem solvers to the configuration of the decision-support system itself. The components of the system are likely to change over time, through modification, replacement, deletion and extension. It should be possible to implement these changes in a seamless fashion through common application programming interfaces and shared databases.

Supportive of these concepts of an open architecture is an object representation that allows the physically interfaced components to cooperate logically through the exchange of information, requests and services. The closer this object representation is to the real world objects that the human decision makers reason about, the more effective the communication among the components and the more intelligent their collaborative assistance will be.

1.4.4 Tools, not Solutions

The decision-support system should be designed as a set of tools rather than as solutions to a predetermined set of problems. The indeterminate nature of complex problems does not allow us to predict, with any degree of certainty, either the specific circumstances of a future problem situation or the precise terms of the solution. Under these circumstances it is far more constructive to provide tools that will extend the capabilities of the human decision-maker in a highly interactive problem-solving environment.

In this sense a tool is defined more broadly than a sequence of algorithms, rules or procedures that are applied largely on the direction of a user. Tools such as software agents can be self-activating, be capable of at least semi-autonomous behavior, and cooperate with each other and users in requesting and providing services. They are employed by each other and users to construct problem solutions to situations that change dynamically and rarely follow predetermined patterns.

1.4.5 High Level Representation

The ability of a decision-support system to have some level of understanding of the meaning of the information it processes is the single most important prerequisite for a collaborative problem-solving environment. A high level representation of the real world objects that define the problem system forms the basis of the interactions between the users and the system and, also, the degree of intelligence that can be embedded in its components. For example, it is virtually impossible to build a useful computer-based tool that can provide meaningful assistance to military commanders in the analysis of the physical battlefield if the battlefield terrain is represented in the computer in terms of 'x,y' coordinates and pixels. To the commander the battlefield consists of real world objects, such as mountains, roads, rivers, trees, observation posts, buildings, and so on. Each of these objects has attributes that determine its behavior under certain conditions. These semantic descriptors form the basis of collaboration among human problem solvers, and are likewise the fundamental unit of communication in a computer-based decision-support environment.

Without an internal high level object representation there can be no partnership between user and computer. Instead the computer is limited to the performance of mundane data processing and visualization tasks, while all of the reasoning activities become the sole province of the human decision makers.

1.4.6 Embedded Knowledge

The decision-support system should be a knowledge-based system. In this context knowledge can be described as experience derived from observation and interpretation of past events or phenomena, and the application of methods to past situations. In other words, we gain knowledge through our experiences with the surrounding environment and our desire to influence our destiny.

Knowledge bases capture this experience in the form of rules, case studies, standard practices, and typical descriptions of objects and object systems that can serve as prototypes. Problem solvers typically manipulate these prototypes, in several different ways (e.g., adaptation, refinement, mutation, analogy, and combination) as they apply them to the solution of current problems (Gero et al. 1988, Rosenman and Gero 1993). Although knowledge is by definition predetermined it plays an important role in human problem solving. New problem situations are rarely if ever, unrelated to past experiences. Therefore, we use our knowledge of past similar situations as a baseline for defining the current problem system and developing a solution strategy.

1.4.7 Decentralized Decision Making

The decision-support system need not, and should not, exercise centralized control over the decision-making environment. Much of the decision-making activity can be localized. For example, components of the system (e.g., mentor agents (also referred to as object agents)) that are responsible for pursuing the interests of real world objects, such as

soldiers in military applications and technical and management personnel in commercial and industrial applications (Pan and Tenenbaum 1991), can achieve many of their objectives through service requests and negotiations that involve only a few nodes of the problem system. This greatly reduces the propensity for the formation of communication bottlenecks and at the same time increases the amount of parallel activity in the system.

The ability to combine in a computer-based decision-support system many types of semi-autonomous and autonomous components (i.e., software agents), representing a wide range of interests and incorporating different kinds of knowledge and capabilities, provides the system with a great deal of versatility and potential for problem solving to occur simultaneously at several levels of granularity. This is similar to human problem solving teams in which individual team members work concurrently on different aspects of the problem and communicate in pairs and small groups as they gather information and explore sub-problems. However, whereas a decision-making environment that includes only human problem solvers is limited to the agent granularity of a single person, the computer-based decision-support system can add to this environment agents that represent elements of the problem system itself.

1.4.8 Emphasis on Conflict Identification

The decision-support system should focus on the identification rather than the automatic resolution of conflicts. This guiding principle gains in importance as the level of complexity of the problem system increases. In very complex problem situations, with many interrelationships, the resolution of even seemingly mundane conflicts can provide subtle opportunities for advancing toward solution objectives. These opportunities are more likely to be recognized by a human decision maker than a computer-based agent.

The identification of conflicts is by no means a trivial undertaking. It includes not only the ability to recognize that a conflict actually exists, but also the determination of the kind of conflict and the relationships that appear to have precipitated the conflict. Tracing these relationships may produce more progress toward a solution than the resolution of the conflict itself. Certainly, automatic resolution of the conflict will greatly diminish the opportunity to explore the conflict situation as a means of gaining a better understanding of the problem system.

1.4.9 The Computer-User Interface

The importance of a high degree of interaction between the user(s) and the various components of the decision-support system is integral to the majority of the guiding principles described in this section. This interaction is facilitated by two system characteristics: a high level object representation; and, an intuitive user-interface. The representation requirements, based on the need for the system to be able to communicate with the user in terms of real world objects that are germane to the problem situation, have already been mentioned in the context of other guiding principles.

There are several facets of the user-interface requirements that should receive attention in the design of a decision-support system. First, the user-interface should be graphical in nature. The human cognitive system excels in pattern matching. Words and numbers require the performance of a translation task that is relatively time consuming, subject to information loss, and carries with it the potential for confusion and misinterpretation. Textual and numerical information should be available to the user on request, whenever a detailed level of precision is desired, but should not be the normal interface mode.

Second, the user should be able to enter instructions and information into the system in a manner that is not tedious. Generally, much headway has been made in this respect in recent years with the introduction of pointing devices and window systems. The selection of options provided by the system, rather than keyboard entry, should extend beyond options and functions to prompted information alternatives automatically extracted from knowledge bases and databases.

Third, an on-line help system should be available to both assist the user in the execution of operational sequences and provide explanations of system activities. The latter should include exploration of the recommendations, evaluation results and proposals contributed by the various components (e.g., agents) of the system. The more sophisticated the capabilities of these components the more important the explanation facilities are to the problem solving environment. In fact, they become a prerequisite for sustaining a productive partnership between the human decision maker and the computer-based assistance components. Furthermore, the on-line help system should be context sensitive. In other words, the help system should provide explanations that are relevant to the specific functional sequence that the user is attempting to perform and for which assistance has been requested.

1.4.10 Functional Integration

In the past it has been considered helpful, as a means of simplifying complex problems, to treat planning and execution as distinct activities. Under this school of thought the purpose of planning is to clearly define and analyze the problem, and then develop a solution as a course of action that can be implemented during the execution stage. However, as the complexity and tempo of problem solving situations increases, these apparently distinct functional areas can no longer be categorized as discrete operational spheres of activity. They tend to merge into a single integrated functional pool of capabilities from which the human decision maker draws assistance as necessary. In such problem solving situations continuous information changes require constant replanning, even during those phases when the need for action and execution overshadows all other activities.

This is particularly apparent in the military field, but equally relevant in management, marketing and manufacturing situations where changing conditions require the most thorough and carefully laid out plans to be spontaneously reformulated. For example, in military missions the impacts of enemy actions dictate the need for continuous replanning

and training during execution. As experienced military commanders are often heard to say *"... there is one thing that is certain under combat conditions, as soon as the first shot has been fired the battle plan is going to change."* Under these conditions functional integration is essential. Not only must the planning functions be accessible from the same computer system, but they must be able to operate on the same information that applies to the execution functions. Much of this information may have been generated in the execution environment and therefore constitutes new information. This in itself dictates the need for replanning, since there is every likelihood that the new information has rendered at least some of the predetermined execution plans obsolete.

Similarly, in the construction and manufacturing fields changing production conditions such as equipment failures and material supply delays may require significant modification of the original design that may border on a complete redesign. These design modifications have to be accomplished while production operations, which cannot be halted or postponed, are in progress. Under these circumstances, design and production functions cannot be treated as discrete operations each with its own set of data, performed in relative isolation from one another. In fact, at that time, the data that describe and are at least partially generated by the production environment are likely to be more relevant to the redesign activities than the original design data. It is therefore of critical importance that the planning and execution environments are logically integrated and driven by the same data flow.

In a distributed, collaborative decision-support system architecture the necessary level of integration has the potential to be achieved, since functional modules and information resources are treated as sharable components. In such a shared environment distributed databases may be accessed by any of the functional components whenever the need arises and the necessary authorizations are available. The ability to switch from one functional mode to another then becomes largely a function of the user-interface and does not require the user to move out of the current application environment. In other words, the physical separation of individual computer-based components need not exist at the logical level of the user-interface.

1.4.11 Learning Capabilities

The ultimate aim of a closely-knit user-computer partnership is to create an environment in which the human and machine contributions are not only complementary, but are also relatively equal in value. For this aim to be achieved one would expect the machine to exhibit at least some semblance of a learning capacity. How realistic is this aim? If we define learning as the ability to develop a problem-solving capacity that is not based solely on predetermined solution methods but evolves progressively from a combination of new and old experiences, then this aim would appear to be achievable. This is particularly true if we allow the user to assist the computer in its learning endeavors, a proposition that is entirely consistent with the notion of partnership.

To qualify under this definition of learning the computer-based components of the decision-support system would need to be able to adapt existing knowledge and apply

the modified knowledge successfully to the solution of problem situations that contain some new elements. However, throughout this endeavor the computer would be able to interact with the user and count on the assistance of the user in two important areas: to interpret the problem situation; and, to select a potentially useful problem-solving strategy. By initially monitoring the actions of the user and storing this second-hand experience, the computer should be able to progressively take an increasingly more active role as a problem-solving partner. It stands to reason that as the computer's knowledge base of combined (i.e., user and computer) experience grows, the ability of the computer-based components to assist the user in the solution of future problems with novel elements should likewise increase. Such an increase would certainly be considered 'learning' in human subjects, and should therefore also qualify as a learning capability in the computer-based environment.

What is being suggested here is that there are categories of learning, and that it is possible with current computer capabilities to achieve some lower levels of learning that depend largely on acquiring existing knowledge rather than creating new knowledge. However, it must be noted that there is no a priori reason that can allow us to assume that lower level learning abilities will eventually evolve into those higher level intuitive learning capabilities that are currently the exclusive domain of human cognition. In other words, whether or not computer-based systems will eventually be able to compete with human decision makers in the realm of intuitive understanding and thinking cannot be predicted at this time.

2. Agents: Thoughts on the Evolution of Computer Software

It can be argued that technical advances in the 20th Century that have led to the availability of electronic computation devices and the interconnection of these devices in information processing networks, are entirely consistent with historical patterns of human evolution. Throughout history an outstanding feature of mankind has been the ability of individuals to leverage their limited capabilities and increase their chances of survival through both technical extensions and cooperative endeavors. For example, instead of facing the inherent uncertainties and dangers of hunting, man developed methods for growing the food that was essential for survival. Success in this endeavor led to the need for cooperation as individuals were able to produce a great deal more food than they required for their own consumption. The storage and distribution of these agricultural products contributed significantly to the development of cooperation, coordination, and planning capabilities in the individual, and increased the propensity for groups of individuals to collaborate toward community goals.

During the Industrial Age, man focused on the manufacture of large quantities of material products that could serve as tools and components for the construction of more complex mechanisms and assemblies such as motors, clocks, rail cars, ships, and buildings. However, by far the majority of these devices and artifacts were aimed directly at extending man's physical capabilities in endeavors such as transporting, building, measuring, fighting, and mining. While this naturally led to significant increases in knowledge and success in intellectual endeavors, the primary objectives were nevertheless focused on material rather than intellectual gains.

It is not surprising that the next major thrust of human endeavor should be directed to computation and information processing. Both of these are cognitive capabilities that appear to be at the core of man's dominant and controlling position among living creatures. Although the initial interests in the development of computer technology were focused on mathematical capabilities, particularly in respect to military and navigational applications (Slater 1989, Kurzweil 1992), this focus soon shifted to the broader spheres of information processing and resource sharing. Today, computer power and capabilities are primarily directed to support man's quest for acquiring and applying information, and only secondarily to the manipulation of numbers and the solution of mathematical equations.

The startling advances in computer hardware that we have seen over the past 25 years have been an essential prerequisite for the emergence of the electronic computer as an extension of the intellectual and cognitive capabilities of the human user. However, the tenfold increase in computer power every five to seven years (Barstow 1987, Kurzweil 1992) has been increasingly distributed to support individual computer users rather than centralized to serve larger groups of users in a shared environment. Microcomputer technology emerged as a means of maximizing local processing power in support of local decision-making. It is interesting to note that the microprocessor has approximately doubled its performance every 18 months, since its invention in the 1970s. In 1996 the

microprocessor became the computer with the fastest clock speed (i.e., 500 MHz), exceeding the clock speed of the Cray supercomputer (Patterson 1996).

The use of the computer as an extension of human cognitive capabilities requires the processing of large quantities of information in the form of databases containing factual information, knowledge bases containing factual data and the relationships among those data, and rule-bases that allow the machine to reason about this knowledge in parallel with the user. To perform such information intensive functions within reasonable response time expectations computer workstations must be able to execute millions of instructions per second in a multi-tasking operational environment and have access to millions of bytes of high-speed random access memory.

The emphasis on information processing has been accompanied by an equally strong desire to share knowledge among widely dispersed users with similar information interests and needs. This is a trend that is currently leading to an unprecedented level of collaboration and coordination among groups of persons, ranging from formal alliances to loosely knit common interest associations. In this networked world of human cooperation the level of sophistication and potential complexity of the interrelationships that can arise spontaneously, or be purposely nurtured by collaborating individuals will challenge the capabilities of the users. Under these circumstances the user will increasingly look to the machine for intelligent assistance. We are, in effect, replacing the computer architecture concept of a single central processor with distributed systems involving a large number of workstations working in parallel. The ability to harness the full capabilities of such loosely coupled information processing configurations depends on the availability of a new generation of system software. The required operating systems need to be as much communication facilitators as they are managers of the widely dispersed heterogeneous computing resources under their control. At the very least they must provide a high degree of reliability in executing the requested processes (e.g., ISIS (Birman and Marzullo 1989)), automatically select the host machines to optimize the computation (e.g., Plan-9 (Pike et al. 1990)), and provide fail-safe inter-process communication facilities (e.g., PVM (Sunderam 1990)).

2.1 The Distributed Cooperative Computing Environment

The initial benefit to the user from computer connectivity has been ready access to information, both in terms of person-to-person communication (i.e., electronic mail systems) and database resources (e.g., libraries, as well as business and government data). In terms of software applications, such as office management, simulation of decision-making sequences, and the automation of design and manufacturing functions, the developments have been less startling. Unfortunately we are still supporting many legacy software systems today that consist of large, single process, computer programs with predefined access paths to their data needs. Considerable difficulties are encountered whenever the output from one program should become the input to another program. These difficulties are related not only to differences in data formats, but are an intrinsic property of the very nature of the underlying program design. It can be argued that these programs are intended to provide solutions to known problems in a prescribed fashion

and are, therefore, designed to accomplish a predefined set of functions in a largely sequential fashion with little or no flexibility for deviating from the predetermined solution path. In other words, they contain ready-made solutions rather then tools that can be employed by the user to develop solutions at the time a problem arises.

The inherent weakness of this approach is the fact that the problems that are of real interest and importance are typically those that were not predicted beforehand or that do not precisely follow the template of the predicted problem model. The outcome of this mismatch depends on the degree of divergence between the actual problem situation and the predefined solution sequence, ranging from a reasonably good result to one that is totally unacceptable. A particularly dangerous aspect of this approach is that it can mislead the user, particularly the novice user, into making decisions on the basis of an erroneous result produced by a computer program that has previously (i.e., under different circumstances) provided useful results.

Human decision makers solve problems through a learning process that depends on cooperation, the ability to assemble lower level knowledge into higher level knowledge, and the freedom to evaluate partial solutions through testing and experimentation. In this sense the real world is a distributed network of information sources, cooperating agents, coordination protocols, and solution strategies. This is a highly interactive environment in which self-determination and cooperation, together, produce achievements that transcend the capabilities of the individual. Distributed computing systems are able to emulate and extend this kind of cooperative problem-solving environment. By linking information and computing resources, by providing means for multiple computer-based agents to interact with each other and human users, and by allowing these activities to occur in parallel, such computer networks can significantly elevate the problem-solving capabilities of human groups and organizations.

However, a new kind of software is required to realize these potential capabilities. Typically, a distributed software application consists of multiple processes, some acting as semi-autonomous agents endowed with capabilities for exercising a degree of self-determination, and others serving as facilitators with largely predefined functional capabilities. The ability to communicate is as important in the distributed computing environment as it is in the real world.

2.1.1 Communication Facilities

Rather than duplicate communication facilities at different software levels within the distributed computing environment it seems appropriate to provide a distributed communication facility that can be used for all purposes. Such a distributed information-serving facility is likely to provide a higher degree of integrity and reliability, or at least provide those features without duplication. In this respect duplication must not be confused with redundancy, which is a function of the communication paths and their control rather than the communication interfaces that are embedded in the applications. Redundancy is, of course, not only desirable but necessary to ensure a high level of reliability.

A shared communication facility will support a number of universal activities, such as the ability to broadcast messages to multiple agents, to identify all agents by asking the facility about its clients, and to monitor and maintain process integrity. More specifically, the communication and event management facilities required in support of a distributed computing environment exist at both the logical and physical levels (Myers et al. 1993). Logically, the underlying communication facility must allow its clients to communicate with each other through object-oriented messages. Clients should be able to use any type of object as an inter-client message without content transformation. Further, client application environments should be able to define and manipulate their own set of objects, both statically and dynamically. Physically, clients should be able to function without knowledge of the operational characteristics of their counterparts (Elmasri and Navathe 1989). Such characteristics include the current state of execution and the physical site location.

Client authorization is another important requirement. Each application should be able to define and manage its own authorization of privileged facilities. This external authorization must work in conjunction with the underlying authentication and security mechanisms provided by the communication system (Pohl 1995). Furthermore, clients should be notified of pending events in real-time (Durfee 1988). This is a critical requirement for providing data validity and currency within the application environment.

2.1.2 Process Management Requirements

Apart from the underlying communication facilities, the distributed computing environment requires substantial process management support. First, users should be able to customize their application environment in terms of both the number and types of agents that will contribute to the solution, and the optimum distribution of those processes on the available machines. This assumes the availability of a high level interface that can intelligently assist the user during the initial application configuration process and, subsequently, whenever modifications are desired (Myers et al. 1993).

Second, there is a need for users to be able to save and restore the current state of their own application environment. This is required not only to allow the user to continue work from previous sessions, but also to experiment with several views of the same problem during a single session. A save and restore facility will also support multi-user work. One user should be able to save the state of the session, allowing other users to utilize this saved state as a starting point for further work. To save the state of multiple processes in a distributed environment, even if these processes are components of the same application, is a fairly complex undertaking. The current state of the environment includes not only the various parts of the system that contain data representations of the evolving solution model and its context, but also the current state of agents (e.g., the local fact list of an expert system). One approach for achieving this is to divide the application system into 'saving' and 'non-saving' processes. All 'saving' processes are required to incorporate save and restore facilities, and 'non-saving' processes are designed to operate as initial invocations whenever they are called. Within the current state of software

technology it would be acceptable to use a mechanism that can identify when no messages are in transmission between components.

Within the next decade, as distributed cooperative systems become the architecture of choice for the majority of integrated decision-support applications, the need to save and restore the current state of an application will increasingly extend to large parts if not the entire system. For example, in the military field it will become highly desirable for commanders to be able to explore the multiple possible outcomes of mission plans with the participation of all of the planning cells that will be found in an Extended Combat Operations Center (ECOC). Today, such scenarios have to be played out in a largely manual mode with limited computer-based assistance available from mostly stand-alone simulation programs in each cell. Once these applications become part of an integrated command, control, communications, computers and intelligence (C4I) system framework commanders will see the opportunity for vastly extended computer war games. While the technical implications of these user expectations are certainly formidable, there is no a priori reason why they could not be achieved. What is required is the ability to save and restore several applications in a distributed system environment. Each of these applications may itself be distributed and is certainly likely to have many cooperative intra-application and inter-system interactions in progress at the time that the current state of the environment is required to be saved.

2.1.3 Explanation Capabilities

As the collaborative decision-support environment becomes more and more capable of providing assistance, through the intelligence and knowledge that is embedded in its agents and other components, there will be an increasing need for the system to be able to explain why it has reached its conclusions. At the very least the agents should be able to explain their behavior and results. In this regard retrospective reasoning is the most common type of explanation facility found in expert systems today. Typically, such a facility is capable of providing answers to WHAT, HOW and WHY questions. A WHAT question requires the explanation or definition of a fact. For example, in a cooperative military decision-support system the commander may ask: What are the mobility capabilities of the enemy unit located at Alpha NJ670900 (where Alpha is a designated sector in the battlefield and NJ670900 indicates the precise location by reference to map grid coordinates)? Through the use of 'format templates' an agent can collect the appropriate answers simply through template values when a match is made with the facts (e.g., enemy, Alpha, NJ670900, mobility) contained in the question (Myers et al. 1993).

A HOW question requires an analysis of the sequence of inferences that produced the fact. Continuing with the above example, the commander may ask: How did the enemy unit reach its present location? A trace of the rule chains that fired to track the movement and transportation capabilities of the enemy unit over time will provide the requested explanation. WHY questions are more complicated. They require reference to the sequence of goals that have driven the sequence of inferences (Ellis 1989). In large

collaborative systems many agents may have contributed to the inference sequence and will need to participate in the formulation of the answer.

This third level of explanation has received considerable attention in recent years. It typically requires a summary of justification components. For example: text summary systems such as Frump (Dejong 1982) and Scisor (Jacobs and Rau 1988); fast categorization techniques such as Construe (Hayes and Weinstein 1991); grammatical inference (Fu and Booth 1975) which allows inductive operators to be applied over the sequences of statements produced from successive justifications (Michalski 1983); explanation-based learning (Mitchell et al. 1991); and, case-based reasoning (Shank 1990 and 1991, Kolodner 1993).

2.1.4 Coordination Strategies and Protocols

By far the most complex and challenging aspect of distributed, cooperative decision-support systems is the control and coordination of the collaborating agents. Not only must each individual agent know how it should most effectively apply its capabilities and local resources, but the agents as a group must also coordinate their activities to maximize the use of network resources. Similarly in human society, interaction among agents can be highly productive or it can be a hindrance to the formation of any kind of useful collaboration. The more complex the interdependencies among problem domains, the greater the potential for cooperation and, conversely, for obstruction.

Protocols of coordination have been studied for more than a decade in the field of distributed artificial intelligence (Chaib-Draa et al. 1992). A popular strategy is to consider a distributed, cooperative decision-support system as a network of problem-solving nodes that individually solve sub-problems and collaborate to integrate narrow solutions into broader solutions. However, to be able to decide how to best collaborate, the nodes require some knowledge of the global goals of the problem-solving system.

Durfee (1988) identifies four principal global goals of cooperation. The first of these goals is to facilitate the completion of problem solution tasks through concurrent activities. Typically, this entails decomposition, development of sub-solutions in parallel, and the application of coordination strategies to minimize the length of time agents have to wait before they receive contributing results from other agents. In addition, this first goal implies the need for a strong emphasis on local problem-solving capabilities. The ability of an individual agent to develop sub-solutions in a semi-autonomous manner, with regard to but not controlled by the concerns of other agents, encourages solution alternatives and communication selectivity.

The second goal focuses on cooperation. It proposes to maximize the potential for task accomplishment through the sharing of resources (i.e., computing resources (CPUs), information, skills, etc.). This goal suggests the need for agents to share predictive information, to exchange tasks, and to assist each other in the testing and evaluation of results. The opportunity for sub-solution verification by agents from several different domains, each with its own knowledge, rules and prototypes, is of particular significance.

The intent of the third goal is to increase the task completion success rate through redundancy and experimentation with alternative solution procedures. Important tasks might be assigned to multiple agents to maintain a high level of reliability even though a local node may have failed. For example, a particular sub-problem might be assigned to several agents that will attempt to solve the same task using different procedures, knowledge, and delegation strategies. A simple application of this goal is the performance of database searches to serve as a basis for providing a common set of information, by multiple agents operating in parallel. As soon as one of the agents has retrieved the required information the concurrent search efforts of the other agents are terminated.

Durfee's fourth goal proposes to minimize the potential for inter-agent interference through the anticipation and avoidance of counter-productive interactions. Of particular importance in this regard are the maintenance of a quasi real-time communication environment, the ability of agents to recognize unnecessary and conflicting tasks, and selectivity in respect to the messages that are sent to other agents or broadcast to the system as a whole.

Conceptually, the protocols that are useful for coordinating a network of problem-solving nodes in a distributed computing environment can be drawn directly from human interactions. However, the implementation of these protocols in a computer-based decision-support environment is a challenging undertaking. For the sake of simplicity we could categorize the interactions of a human problem-solving team into two groups: advisory services that are provided by individual team members in response to requests for knowledge and opinions that fall into their domain of expertise; and, actions of team members that are motivated by personal and global problem-solving objectives.

Past implementations of multi-agent decision-support systems have focused primarily on the first category of interactions. In these systems agents typically represent narrow domains of expertise and are designed to react to requests for services pertaining to these domains. For example, a Weather agent capable of analyzing changes in weather conditions would be called upon by other agents or users to provide weather forecasts and interpret the likely impact on a given situation of specific weather related factors. Cast in this role as a domain expert the Weather agent has no interest in promoting the importance of weather considerations in the overall development of the problem solution. That issue, which may be of paramount importance to the final outcome, becomes the responsibility of the conflict resolution components of the system. At that level the 'weather element' has lost its autonomy and becomes just another issue that will be weighed with other factors as the system attempts to reach a consensus.

Although such systems constitute a significant advance over fragmented simulation-type decision-support systems, they nevertheless represent only a limited implementation of the human interaction model. They place a considerable burden on those system components that are required to coordinate the sub-solutions that are received from the various domain experts. By and large these coordination components have to rely on mathematical weighting schemes and user interaction to resolve conflicting responses.

Furthermore, once a decision has been made, future reevaluations of that decision depend almost entirely on two factors: largely predetermined criteria for triggering the reevaluation of a past decision that are embedded in the coordination components; and, the ability of the user to recognize the need for reconsideration. What is clearly missing in this design approach is the ability of the system to actively represent the interests of individual problem domains even after a conflict resolution decision has been made; namely, the second category of human problem-solving team interactions.

One promising, emerging solution to this dilemma is to extend agent status beyond knowledge domains (i.e., service agents) to factors and/or issues that are of fundamental importance to the problem situation (Pohl et al. 1994). In this approach selected data objects (e.g., mentor agents) such as soldiers in the battlefield, major components in engineering design, or desirable solution attributes (e.g., stealth, speed, safety, economy) are represented in the system by agents. These object-agents, endowed with communication capabilities and knowledge of their own needs and objectives, are capable of actively promoting their interests at any stage in the problem-solving process. Of particular interest in this regard is the ability of such object-agents to request services from domain experts, negotiate with each other, and interact with users. A prototype system of this kind is described later in Section 9 of this report (Pohl 1996).

2.2 The Human-Computer Partnership

To look upon decision-support systems as partnerships between users and computers, in preference to automation, appears to be a sound approach for at least two reasons. First, the ability of the computer-based components to interact with the user overcomes many of the difficulties, such as representation and the validation of knowledge, that continue to plague the field of machine learning (Forsyth 1989, Thornton 1992, Johnson-Laird 1993).

Second, human and computer capabilities are in many respects complementary (Figures 1 and 2). Human capabilities are particularly strong in areas such as communication, symbolic reasoning, conceptualization, learning, and intuition. We are able to store and adapt experience and quickly grasp the overall picture of even fairly chaotic situations. Our ability to match patterns is applicable not only to visual stimuli but also to abstract concepts and intuitive notions. However, although the biological bases of our cognitive abilities are massively parallel, our conscious reasoning capabilities are essentially sequential. Therefore, human decision makers are easily overwhelmed by large volumes of information and multi-faceted decision contexts.

As human beings we have great difficulty dealing with more than two or three variables at any one time, if there are multiple relationships present. Under these circumstances we tend to switch from an analysis mode to an intuitive mode in which we have to rely almost completely on our ability to develop situation awareness through abstraction and conceptualization. While this is our greatest strength it is also potentially our greatest weakness. At this intuitive meta-level we become increasingly vulnerable to emotional influences that are an intrinsic part of our human nature and therefore largely beyond our control.

This emotional dependency also tends to make the human being somewhat unpredictable and strongly resistant to change. Confidence in our ability to deal with complex and critical situations is based to a large extent on our past experience of similar problem situations. Change, by necessity, devalues this past experience and therefore reduces our confidence to be able to successfully deal with the changed situation.

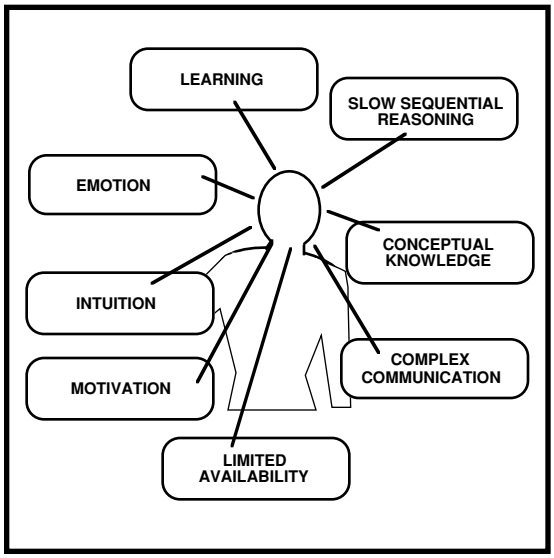


Figure 1: Human Abilities and Limitations

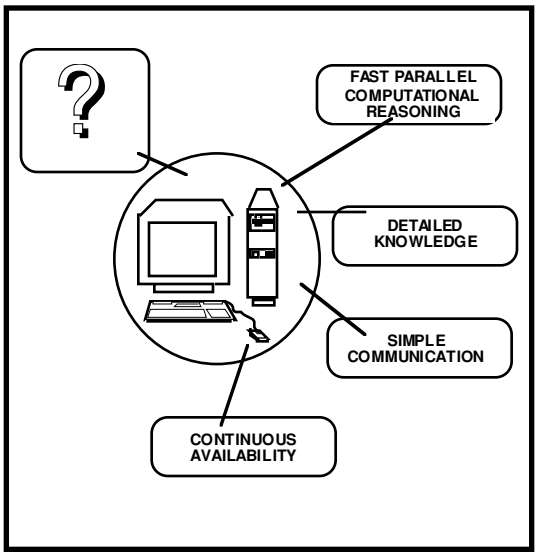


Figure 2: Computer Abilities and Limitations

The capabilities of the computer are strongest in the areas of parallelism, speed and accuracy (Figure 2). Whereas the human being tends to limit the amount of detailed knowledge by continuously abstracting information to a higher level of understanding, the computer excels in its almost unlimited capacity for storing data. While the human being is prone to making minor mistakes in arithmetic and reading, the computer is always accurate. A slight diversion may be sufficient to disrupt our attention to the degree that we incorrectly add or subtract two numbers. However, if the error is large we are likely to notice that something is wrong further downstream due to our ability to apply conceptual checks and balances. The computer, on the other hand, cannot of its own accord (i.e., at the hardware level) distinguish between a minor mistake and a major error. Both are a malfunction of the entirely predictable behavior of its electronic components.

However, at the software (application) level it is possible to provide a layer of automatic reasoning capabilities (i.e., collaborating agents) served by an underlying information model (i.e., ontology). Software with such embedded capabilities is able to draw inferences leading to more sophisticated human-like conclusions.

The differences between the human being and the computer are fundamental. All of the capabilities of the digital computer are derived from the simple building blocks of '0' and '1'. There is no degree of vagueness here, '0' and '1' are precise digital entities and very different from the massively parallel and largely unpredictable interactions of neurons and synapses that drive human behavior. It is not intuitively obvious how to create the high level representations of real world objects (e.g., ship, aircraft, dog, house, power, security,

etc.) that appear to be a prerequisite for reasoning and learning capabilities, in a digital computer. While these objects can be fairly easily represented in the computer as superficial visual images (in the case of physical objects such as aircraft, weapons and buildings) and data relationships (in the case of conceptual objects such as power and security) that in itself does not ensure that the computer has any understanding of their real world meaning. These representations are simply combinations of the basic digital building blocks that model, at best, the external shell rather than the internal meaning of the object.

Unfortunately, it is still not generally understood that this representational inadequacy is the single most limiting factor in virtually all existing decision-support systems. For example, current military command and control systems tend to overwhelm commanders with hundreds of detailed satellite pictures of battlefield conditions that are transmitted by computers as digital packages rather than groups of objects. As a result the interpretation, filtering and fusion of these images, areas in which computer-assistance would be highly desirable, become the burdensome task of the human decision maker.

More than 10 years ago when the Collaborative Agent Design Research Center (CADRC) first embarked on the development of cooperative multi-agent systems we recognized the fundamental importance of representation, as a prerequisite for providing computer-based agents with reasoning capabilities. We discovered that while this problem was well known and had been the subject of considerable research in the artificial intelligence community, the results of this research work had generally remained the province of that close-knit community.

Early practical implementations of artificial intelligence systems were almost exclusively confined to stand-alone applications, such as expert systems (e.g., Prospector (Duda et al. 1977, Reboh 1981), MYCIN (Buchanan and Shortliffe 1984), and ASTA (Wilson et al. 1984)). Since these systems were not intended to interface with other applications the importance of representation continued to be largely ignored by the mainstream of software developers and users. Over the past decade the CADRC has explored, adapted and implemented several high level representation techniques in its various decision-support applications for industry and government sponsors (Myers et al. 1993). While there is a need for a great deal more work in this area the state of technology today is, without question, capable of providing an internal representation level that can support meaningful reasoning assistance in large integrated decision-support systems.

2.3 Computer Applications as Agent-Based Systems

In the broadest sense an agent may be described as a computer-based program or module of a program that has communication capabilities to external entities and can perform some useful tasks. According to this definition agent software can range from the simplest, stand-alone, predetermined, algorithmic application to the most intelligent, integrated, multi-agent decision-support system that advanced technology can produce today. While such a broad definition may afford little insight into the nuances of agent characteristics and behavior, it does provide a convenient basis for briefly tracing the

evolution of computer applications (Figure 3) within three 'waves' of software development, namely: stand-alone, single-agent software that is mostly procedural in nature; multi-agent and object-agent software with cooperative and collaborative capabilities, respectively; and, software that supports adaptive and emergent knowledge systems .

2.3.1 '1st Wave' Applications Software

Typically, 1st Wave computer applications are large single-process programs that execute in a sequential fashion with limited user interaction (Figure 4). Data enter these programs mostly at the beginning of the step-by-step execution path and results become available at the end. The inability of the user to observe the progressively evolving solution and, if desirable, redirect the execution results by changing the values of parameters mid-stream is a particularly cumbersome feature of this type of software. Similarly, it is exceedingly difficult to link these programs with other programs into larger software systems, so that the results generated by one program can be used to influence the results of another program. 1st Wave software was simply not designed to communicate within a collaborative environment. Each program is intended to run to completion before sharing its results with the world. In the case of deep simulations this means that much of the execution time may be wasted, since the user is unable to monitor the developing solution and halt or redirect the simulation if it is clearly moving in the wrong direction.

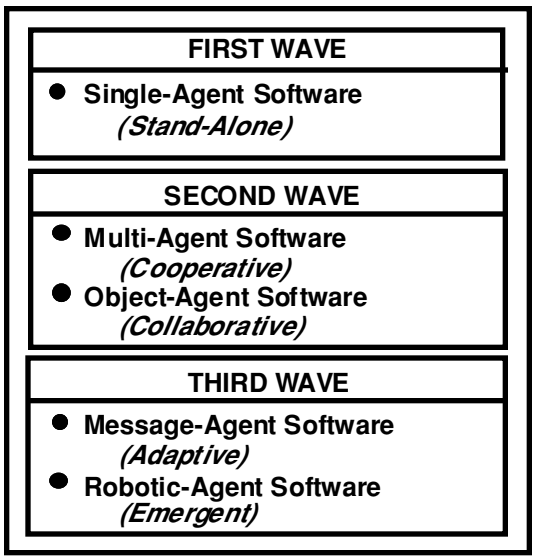


Figure3: Computer Software Evolution

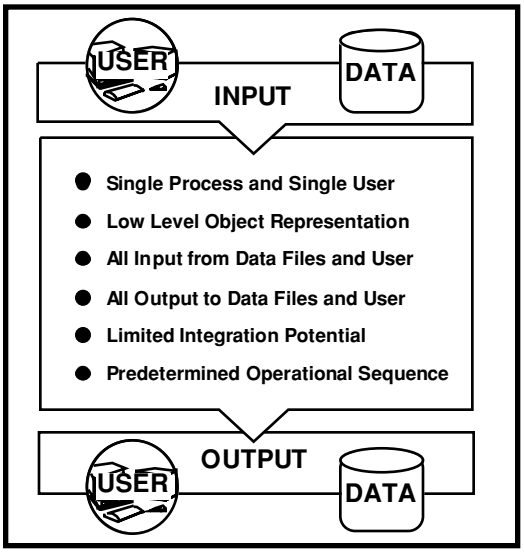


Figure 4: '1st Wave' Computer Applications

An equally serious shortcoming of 1st Wave software is that it typically has no understanding of the nature of the information that is being processed. It recognizes data only at their lowest level of representation, such as text strings, numbers and coordinates, rather than objects with characteristics and relationships to other objects. This not only exacerbates the integration problem, but also renders any efforts to add reasoning capabilities to these programs hardly worthwhile.

1st Wave software assumes that decision-making is essentially a sequential process in which every subsequent step depends on the completion of the preceding step. This view of decision-making is far removed from real world experience, where project teams solve problems collaboratively and contribute to the decision-making process whenever they have something useful to share. Seldom, if ever, is a team member prevented from contributing information until a certain stage or milestone has been reached. On the contrary, team members are encouraged to exchange information freely in the hope that their contributions will accelerate the solution process and increase the quality of the decisions that are made.

2.3.2 '2nd Wave' Applications Software

Adaptation of 1st Wave software to increasingly more complex real world problem situations has led to a hybrid of human and computer-based decision-support systems (Figure 5). Individual members of the human problem team utilize computer-based tools to assist them mostly with the computational and planning components of their tasks. However, this assistance is limited to the individual team member. While the computer can retrieve and send information from and to shared databases, it exercises these capabilities only on the request of its user. Collaboration within the problem team is largely restricted to the communications initiated by team members. The computer shares in these communications only to the extent that its user initiates queries to shared databases. The computer functions as a stand-alone agent that interacts with its user, but does not actively participate in the collaborative problem solving process.

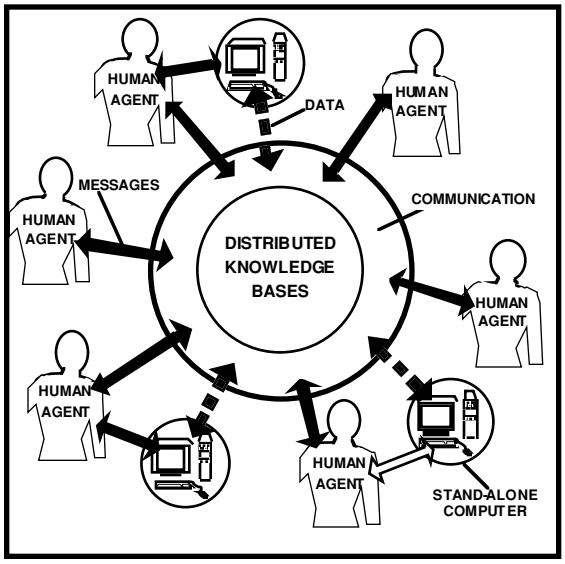


Figure 5: Limited Computer Assistance

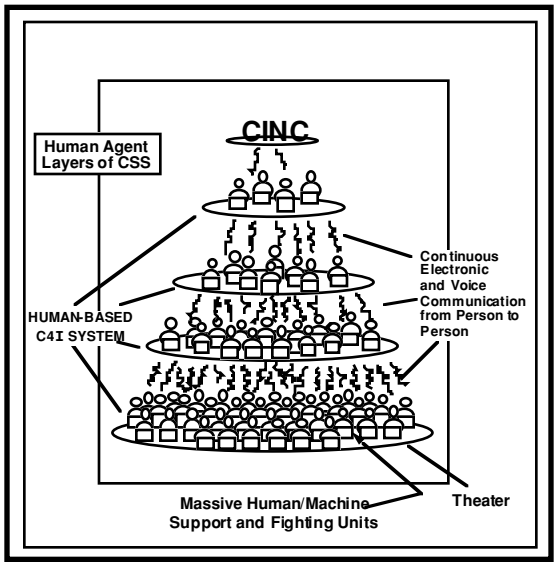


Figure 6: Hierarchical Military C4I Structure

In this hybrid decision-support environment, which is still representative even of the more critical transportation and military systems today, much of the collaboration is based on human-to-human voice communication. As a result, under severe stress conditions (i.e., surge conditions) these systems are subject to serious communication

bottlenecks that will disrupt and may even terminate the decision-making process. In recent years examples of these conditions have occurred during environmental disasters, such as earthquakes in the US, and military missions, such as Desert Storm in the Middle East. In the latter case, as shown in Figure 6, the combination of a hierarchical command and control structure with a 1st Wave software architecture produced a high potential for communication failure. A massive build-up of US and allied forces (i.e., more than 500,000 personnel) in the theater was supported by computer-based communication facilities that reflected the chain of command through multiple levels from the commander in chief (CINC) down to the soldier in the battlefield. In this human-based C4I system environment continuous electronic and voice communication, essentially from person-to-person, quickly clogged the available communication channels.

During the late 1990s the limited computer-assistance capabilities (Figure 5) that are reflective of 1st Wave software will be increasingly replaced by integrated, multi-agent, cooperative systems. This signals the emergence of 2nd Wave software (Figure 7) in which the contributions of several decision-support components are coordinated through an object-serving inter-process communication facility. The components, commonly referred to as agents, may be separate processes or modules of one or more processes. They may be rule-based expert systems, procedural programs, neural networks, or even sensing devices. Increasingly, these agents will have the ability to explain their actions and proposals, as they interact spontaneously with each other either directly or through coordination facilities.

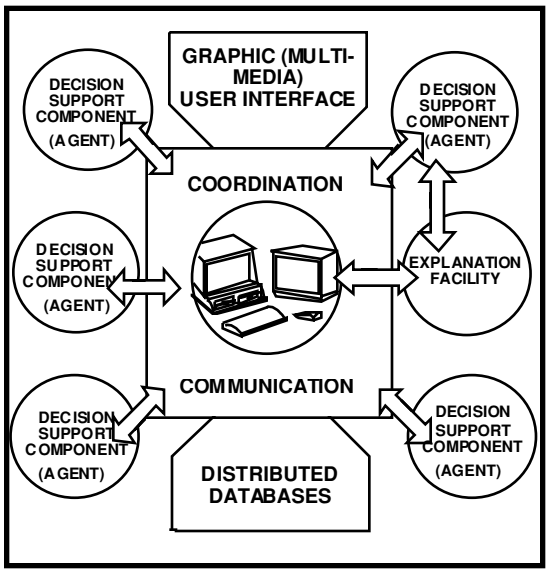


Figure 7: 2nd Wave Computer Applications

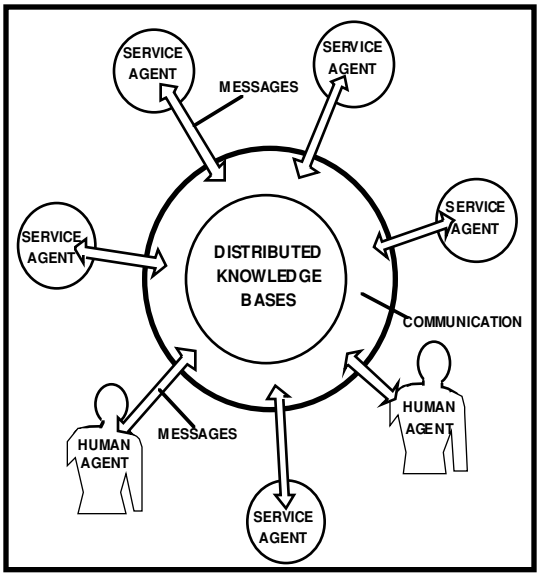


Figure 8: The Service-Agent Architecture

As discussed previously, 2nd Wave software requires a high level internal representation of the real world objects that are central to the problem situation. This is a prerequisite for the reasoning capabilities of the agents and also for the interaction of the user(s) with the system. The objective of 2nd Wave software is not to automate the decision-making activity, but to create an effective partnership between the human decision maker and the computer-based agents. In this partnership the human agent must be able to communicate

with the computer-based agents in terms of the same real world objects that are used so effectively in all human reasoning endeavors. In their role as active collaborators the computer-based agents will have information needs that cannot be totally predetermined. Therefore, similar to the human agent, they will require the capability to dynamically generate database queries and initiate user interactions. At least some of the information sources accessed by the agents will be prototypical in nature (i.e., standard practices, case studies, and other typical knowledge pertaining to the problem situation), consistent with the notion of knowledge-based systems.

As mentioned previously (see Section 2.2), human and computer capabilities are complementary in many respects. Where we human beings excel in the areas of abstraction, conceptualization, intuition and creativity, the performance of the computer cannot be described as being even adequate. However, when it comes to computational speed and accuracy, searching for and storing data, redundancy and parallelism, information persistence, and continuous availability, the computer outperforms us by far. It is therefore not surprising that current, 2nd Wave software, developments are increasingly focusing on collaborative systems in which users interact with computer-based expert agents (Figure 7). Typically, each agent is designed to be knowledgeable in a narrow domain, and represents the viewpoint of that domain in its collaborative endeavors. In this respect it provides services and can be categorized as a service-agent (Figure 8).

The service-agents are endowed with a communication facility that allows them to receive and send information. The manner in which they participate in the decision-making activities depends on the nature of the application. They can be designed to respond to changes in the problem state spontaneously, through their ability to monitor information changes and respond opportunistically, or information may be passed to them in some chronological order based on time-stamped events or predefined priorities. They should be able to generate queries dynamically and access databases automatically whenever the need arises. In other words, service-agents should have the same data search initiation capabilities as the user and should not be dependent solely on the user for access to external information sources. In fact, the human users in such multi-agent systems may be categorized as very intelligent, multi-domain service agents. Examples of such service-agent systems can be found in the literature (Durfee 1988, Pohl et al. 1989, Lesser 1995).

Within a networked environment the service-agents pertaining to a single multi-agent system (Figure 8) may be distributed over several computers, and even the coordination facilities (i.e., planning, negotiation, conflict detection, etc.) may be distributed over several nodes (Pohl et al. 1992). Alternatively, several single multi-agent systems can be connected (Figure 9). In this case each multi-agent system functions as an agent in a higher level multi-agent system. Such systems are well suited to planning functions in which resources and viewpoints from several organizational entities must be coordinated.

Typical application areas include military mission planning and facilities management. The user at each node should be able to plan in multiple worlds. For example, a private world in which shared information sources may be accessed but the deliberations of the user are not shared with other users, and a shared world that allows and encourages the

continuous exchange of comments, plans and instructions. The capability normally exists for the user to maintain multiple views of each world to facilitate experimentation and the exploration of alternatives (Nadendla and Davis 1995). The service-agents resident in each system (i.e., at each node) should be able to differentiate between worlds and also between the views of any particular world. This will require a high degree of parallelism that must be supported by the system architecture.

Multiple service-agent systems offer many opportunities for application customization. For example, the viewpoints and objectives of a particular organizational entity or user can be represented by the specific combination of service-agents and the capabilities and priorities of each individual agent. In the case of a facilities management application involving real estate properties, the often distinctly varying objectives of the building owner, the needs of the different occupant groups, the security staff, and the maintenance personnel, will require a significantly different mix of agent service capabilities at each node. Also, each node will require access to databases that are shared by all or several nodes and distinct information sources that are only locally relevant. The cooperative decision-support capabilities of these distributed systems extend beyond the assistance provided by the local service-agents, to the ability of users and software agents to negotiate both through direct communication and by sending partial or complete solution proposals between nodes.

So far we have discussed multi-agent systems involving two types of agents; namely, service-agents and human agents (i.e., users). Other agent types are certainly feasible. Of particular interest is the agentification of the information objects that are intrinsic to the nature of each application. These are the information objects that human decision makers reason about, and that constitute the building blocks of the real world representation of the problem situation. The fundamental need for the computer-based decision-support system to share this high level representation with the user has been discussed previously.

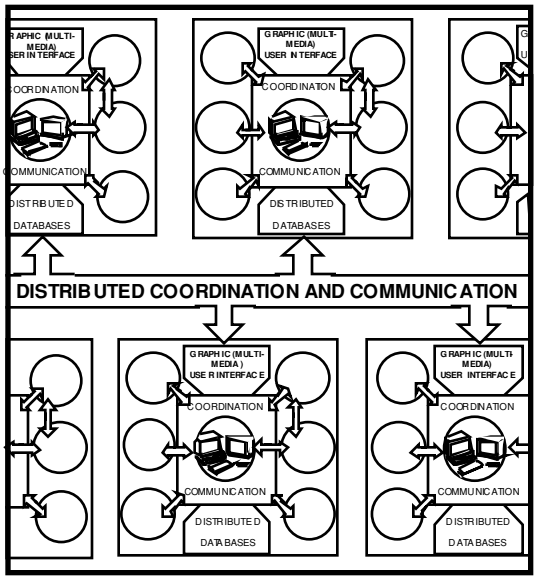


Figure 9: Connected Service-Agent Systems

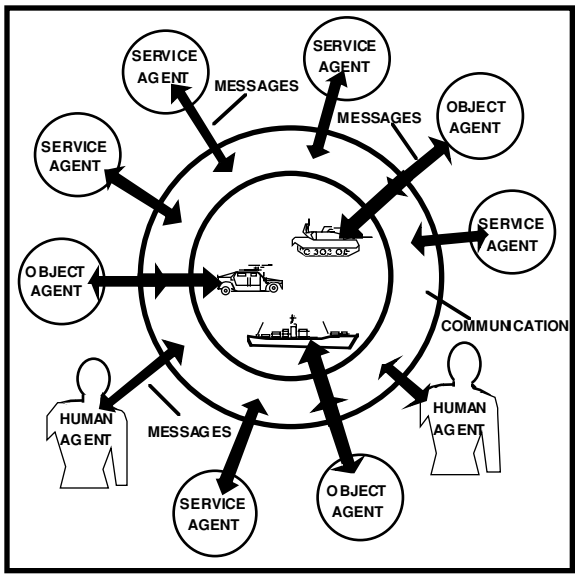


Figure 10: Object-Agent Systems

The notion of object-agents brings several potential benefits. First, it increases the granularity of the active participants in the decision-making environment. As agents with communication capabilities, objects such as armored vehicles (in military missions), aircraft (in air traffic control), or building spaces (in architectural design), can pursue their own needs and perform a great deal of local problem solving without continuously impacting the communication and coordination facilities utilized by the higher level components of the decision-support system (Figure 10). Typically, an object-agent is a process (i.e., program) or component of a process that includes several adjuncts that provide the agent with communication capabilities, process management capabilities, information about its own nature, global objectives, and some focused problem solving tools.

Second, the ability of object-agents to request services through their communication facilities greatly increases the potential for concurrent activities. Multiple object-agents can request the same or different services simultaneously. If necessary, service-agents responding to multiple service requests can temporarily clone themselves so that the requests can be processed in parallel. Third, groups of object-agents can negotiate among themselves in the case of matters that do not directly affect other higher level components or as a means of developing alternatives for consideration by higher level components. Fourth, by virtue of their communication facilities object-agents are able to maintain their relationships to other objects even though they are themselves a product of 'decomposition'. In other words, the concept of object-agents overcomes one of the most serious deficiencies of the rationalistic approach to problem solving; namely, the dilution and loss of relationships that occurs when a complex problem is decomposed into sub-problems. In fact, the relationships are greatly strengthened because they become active communication channels that can be dynamically created and terminated in response to the changing state of the problem situation.

2.3.3 '3rd Wave' Applications Software

The combination of object-agents and service-agents in the same decision-support system suggests a logical transition from 2nd Wave to 3rd Wave software in which even simple learning capabilities may eventually lead to emergent knowledge (Brooks 1990). Object-Agents may represent abstract concepts such as image and power, collective notions such as climate, virtual entities such as a building space during the design process (Pohl 1996), physical objects such as a M1A1 tank in the battlefield, or even human beings such as an individual soldier, squad or platoon. In the latter case a small communication device, embedded in a computer tag, is attached to the uniform of the soldier. This Radio Frequency Tag (RF-Tag) is capable of receiving and sending messages to an object-agent taking the role of a mentor within the computer-based command and control system. In this scenario the object-agent can serve many functions. It can provide several kinds of assistance to the soldier, such as medical advice, geographical position and terrain information, enemy location and strength, maneuver strategies, fire support alternatives, and so on. Conversely, the object-agent can use the soldier as part of a sensory array that continuously collects intelligence with and without the soldier's direct involvement.

Many of the service requests received by the object-agent will need to be passed onto service-agents, human agents, or other object-agents (Figure 11). This can be accomplished through the appropriate use of both broadcasting and directed modes of communication. For example, a request for medical advice may initiate several actions by the mentor agent: a specific request for more detailed information to the soldier; the collection of bodily functions data from sensors embedded in the soldier's uniform, if the soldier has been wounded; a broadcast for evacuation assistance, if the wounds are serious; a request for specific self-help medical advice directed to a service-agent with medical expertise; a situation update to the commander's mentor agent and/or the designated command and control service-agent; and so on. Even if the soldier is unable to personally communicate, the mentor agent is automatically alerted to the soldier's medical condition through sensors attached to his or her uniform or skin.

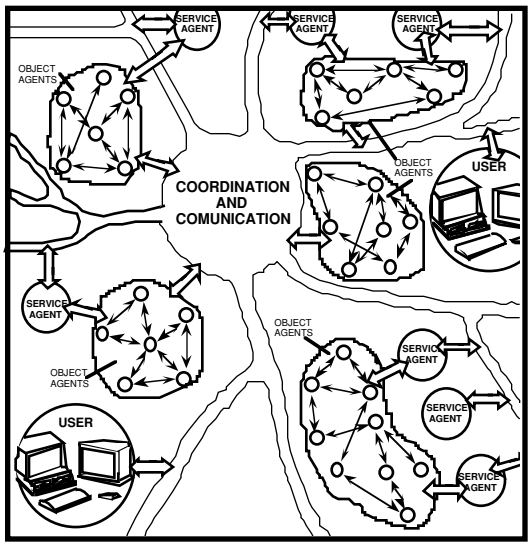


Figure 11: Object-Agents and Service-Agents

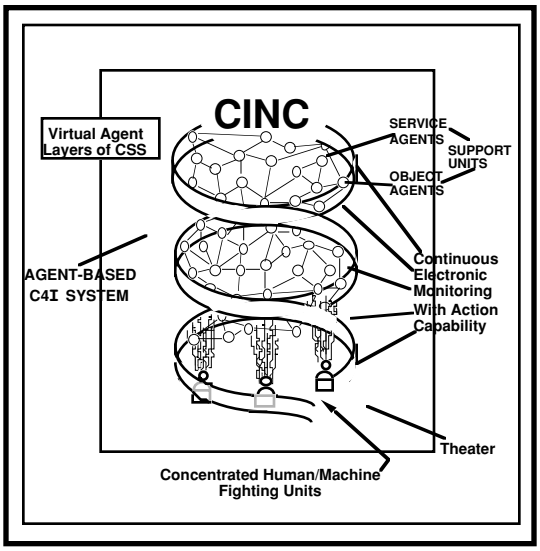


Figure 12: Cooperative Military Command and Control System

The schematic representation of an agent-based cooperative command and control system shown in Figure 12, differs from the conventional 'human-based' command and control system shown previously in (Figure 6) in many significant respects. First, the continuous and automatic monitoring of human/machine fighting units by the various types of agents that operate spontaneously within the communication system provides the warfighter with access to instantaneous advice and guidance. The agent-to-agent communication that facilitates this continuous access to information and intelligent analysis is not dependent on human-to-human interaction. In a conventional command and control system the communication channels are easily saturated by the continuous flow of human-to-human electronic and voice communications. Efforts to control this traffic inevitably require the imposition of communication restrictions that can easily prevent critical information from reaching the appropriate commander or warfighter. In addition, as was shown in Figure 6, the human-to-human interaction encourages a build-up of support personnel in and around the theater. This build-up is costly in terms of

transportation and logistics, increases the danger of casualties, and places an additional burden on the already overloaded communication facilities.

Second, the multi-agent system architecture decentralizes both the collection and analysis of information. Individual human and machine warfighting units serve equally well as collectors and generators of data, as they do as recipients of data. In this way a dispersed force of warfighters can represent an important data sensor array, with the ability to add value by converting the data into information and knowledge close to the source. This decentralization of the data analysis process is particularly valuable in terms of distributing the communication traffic and validating the results of the analysis at the collection source.

Third, the seamless integration of planning, execution and training functions within the same command and control communication system allows the commander and the individual warfighter to continuously and instantaneously switch from one mode of operation to another. In fact, the parallel nature of the system will allow specific planning, execution and training tasks to be undertaken concurrently. For example, the commander may wish to initiate a planning function through one set of agents while executing a specific operation in the theater, and at the same time simulate a particular 'what if ' scenario in anticipation of a possible future situation.

Recent studies by the US Marine Corps and the US Army have demonstrated the capabilities of relatively low cost computerized RF-Tags that are mounted on vehicular cargo. Object-Agents can be designed to communicate with tagged equipment not only for purposes of monitoring their location, but also in a service and low level decision-making role. For example, let us assume a tactical cargo load-planning scenario in which a fuel truck, fitted with a RF-Tag has been loaded onto a ship. During the voyage the fuel truck starts to leak. While the volume of fuel leaked is fairly small, even this small amount constitutes a serious potential hazard onboard ship. Alerted of the situation through a simple feedback mechanism the RF-Tag communicates to its companion object-agent, resident in the command and control system, both its location and the extent of the leakage. The object-agent analyses the situation, either through its own capabilities or by requesting supporting services from other agents, and automatically notifies appropriate command personnel, or other agents, or the ship directly. What is particularly noteworthy in this scenario is the fact that the command and control system was not only able to automatically detect the problem, but also analyze the situation and take action without the need for human intervention.

In existing multi-agent system configurations that include only domain agents (i.e., service-agents), conflicts arise when agents either disagree among themselves or with a decision made by the user. For example, utilizing such a system for the load-planning of a ship, the placement of a fuel truck in a particular ship compartment might provoke the latter type of conflict. If the stow-planner unknowingly places the truck in the immediate vicinity of another cargo item of a different hazardous material class, then the Hazard agent will alert the user and explain the necessary segregation requirements. The stow-planner resolves the conflict by relocating or unloading one or both of the cargo items or,

alternatively, overrules the service-agent. The fuel truck, as a passive object, is involved in the conflict resolution process only as an information source that is used by the service-agent in its deliberations. In other words, while the validation of the load-planning decision is entirely dependent on the knowledge encapsulated in the object the latter is unable to actively participate in the determination of its own destiny.

The situation is somewhat analogous to a scenario common in real life when one or more persons feel compelled to make decisions for another person, although the latter might be more competent to make those decisions himself. The outcome is often unsatisfactory because the decision makers tend to use general domain information where they lack specific knowledge of the other person. The 'individuality' of the problem situation has been usurped by the application of generalizations and, as a result, the quality of the decisions that have been reached are likely to be compromised. In the example of the two hazardous cargo items, if the fuel truck were to be represented by an object-agent then much of the decision-making could be localized within the knowledge domain of the agent. As soon as the fuel truck has been placed in the ship compartment by the stow-planner the Fuel Truck agent could broadcast two specific requests for service: Where am I located?; and, What are the locations of other hazardous cargo items? The answers to these questions can be compared by the Fuel Truck agent directly to what it knows about its own mobility and access capabilities. The development of alternative strategies for resolving the hazardous material problem can now take place within the context of all of the information in the Fuel Truck agent's knowledge domain. For example, the possibility of relocating itself to another compartment that already contains hazardous material of the same class can be explored by the agent (with or without the active collaboration of the stow-planner) as a direct consequence of its own deliberations. In multi-agent systems incorporating only service-agents this remedy is less likely to be proposed, because the interests of the fuel truck are fragmented among the various service-agents that drive the conflict resolution process.

There is another kind of conflict resolution scenario that becomes possible with the availability of object-agents. An object-agent may develop a solution to a sub-problem in its own domain that redirects the entire course of the overall solution plan. For example a squad, operating in dispersed mode in enemy territory and communicating with a mentor agent (Figure 13), performs its assigned enemy surveillance mission. It communicates through its object-agent certain enemy behavior that it believes could be turned to advantage if specific elements of the current overall operations plan were to be modified. However, such suggestions are rejected at operational levels below the commander for reasons that appear to this squad to be based on erroneous intelligence. The squad judges the matter to be of a potentially serious nature and instructs its mentor agent to validate aspects of the squad's current understanding of the battlefield situation.

The object-agent commences a low level investigation by communicating with the mentor agents of several other squads and utilizing the services of domain agents (i.e., service-agents) where necessary. Soon an alarming picture emerges. It appears possible that the enemy has infiltrated one node of the command and control system and is entering erroneous data through this node. The effects of this gradually evolving deception could

lead to disastrous consequences. The squad, realizing the potentially serious nature of the situation, progressively develops through the activities of its object-agent a more and more compelling case in support of its observations and suggestions. Eventually, the overwhelming weight of evidence developed from the interactions of the squad with its object-agent and other agents in the command and control system attracts the attention of the Command Element. The commander and his object-agent quickly undertake another analysis of the situation considering additional factors not considered in the squad's analysis. He verifies an almost certain localized penetration by the enemy of the command and control system and decides to utilize this knowledge by implementing a double-deception strategy.

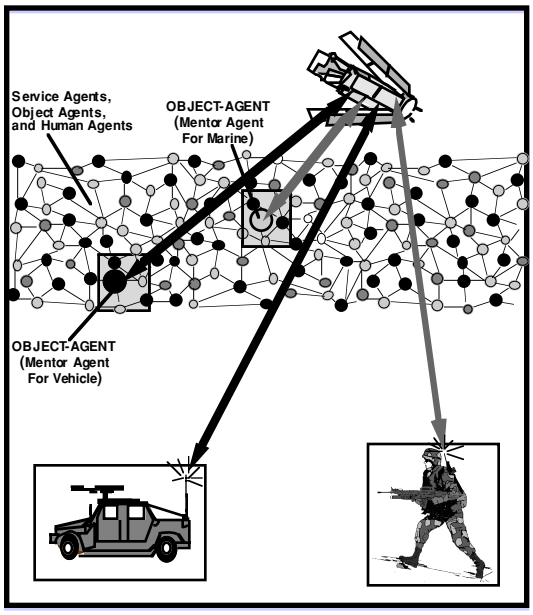


Figure 13: Object-Agents as Mentors in an Integrated Command and Control System

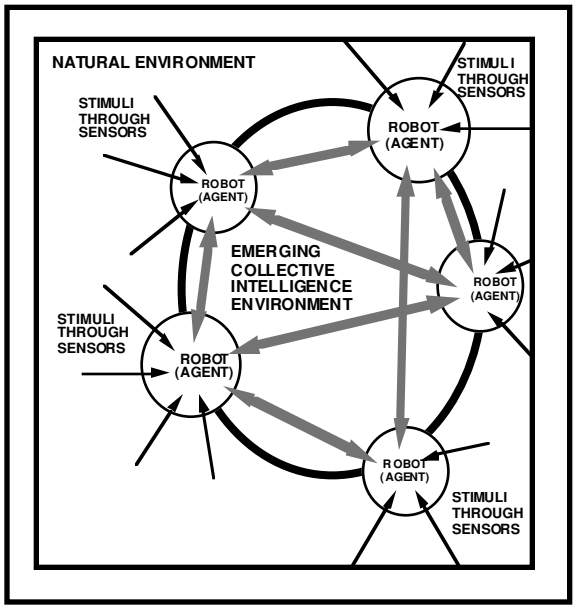


Figure 14: 3rd Wave Applications Software

This scenario demonstrates several significant capabilities of a multi-agent command and control system incorporating object-agents. First, it is significant that the likely enemy penetration of the information system has been discovered at all. If the squad had been restricted to communicating its data as passive objects for processing by service-agents there would not have been any desire on the part of the command and control system to pursue the problem after the initial conflict resolution. Second, the squad's object-agent was able to undertake its investigation in a decentralized fashion without impacting higher level command and control activities until it was ready to present a strong case for reconsideration. However, it was able at any time to alert higher levels of the command structure as soon as the results of its investigation warranted such action.

Third, if the squad's projections had been rejected at all higher agent levels, the squad's object-agent could have appealed directly to the commander or his object-agent. Under these circumstances the commander would have several alternative courses of actions open: also reject the squad's suggestions; require one or more of the higher level agents

(i.e., object-agents and service-agents) to explain their ruling; reset certain parameters that would allow the higher level agents to reconsider their ruling; overrule the higher level agents and accept the proposal; or, capture the current state of the battlefield situation as a recoverable view and use the squad's proposition as the basis for the exploration of alternative solution paths.

Apart from their immediate action capabilities, object-agents support the highly desirable goal of decentralization through localized decision-making and communication. In this kind of distributed, cooperative environment it would be useful if messages themselves could be endowed with agent capabilities. At least certain types of messages would benefit greatly from action capabilities. For example, a message-agent sent by an object-agent or service-agent to find particular information could clone itself to seek the information concurrently in several potential sources. Once apparently relevant information has been found it could be synthesized to formulate a meaningful response to the originator of the query. Clearly, message-agents would add another level of granularity, decentralization and action capability within the distributed, cooperative decision-support system architecture.

The nature and typical mechanisms of the four agent types discussed may be summarized in terms of the interests they represent and the mechanisms they utilize to reach the objects that are derived from these interests.

- ***Human Agents*** (i.e., users) represent intelligent life and their typical mechanisms include: continuous sensory input (i.e., sight, touch, smell, taste, and hearing); complex communication in the form of speech, body language, and extrasensory perception; and, the ability to create knowledge through learning and intuition.

- ***Service-Agents*** represent domain expertise and their typical mechanisms include: simple communication in the form of limited vocabulary messages and limited sensory data; logical inferencing based mostly on the application of deductive and some inductive techniques (Shapiro 1987); and, the performance of services within the narrow boundaries of their domain knowledge.

- ***Object-Agents*** represent data entities and their typical mechanisms include: simple communication capabilities and logical inferencing skills on par with those of the service-agent; the ability to collaborate and negotiate with other agents as they seek to satisfy their needs and objectives; and, the ability to request specific services from service-agents and human agents.

- ***Message-Agents*** represent lower level actions that are intended to contribute to higher level decision processes. Their typical mechanisms include: the creation of clones of themselves to initiate the same action along multiple paths; the synthesis of multi-source responses; and, triggering of actions at their destination.

These developments in distributed, cooperative, multi-agent software architectures are converging in several respects with another approach to the design and implementation of

intelligent decision-support systems. This approach is based on the concepts of emergent knowledge. Whereas the multi-agent systems that characterize 2nd Wave software depend largely on predefined knowledge and high level problem solving processes, 3rd Wave software (Figure 14) builds on agents with low level skills that cooperatively and cumulatively develop an emergent collective intelligence.

A simplistic analogy can be made by comparing the knowledge acquisition and learning behavior of an infant with the established knowledge base of an adult. Infants learn about their world through environmental stimuli that are processed through their motor-sensory facilities and stored as experience. Progressively, this experience is conceptualized and abstracted into higher level knowledge as the infant moves through childhood into adulthood. The essence of this human evolution is a continuous process of learning and adaptation that is based on the interactions and contributions of a multitude of low level capabilities, none of which on its own can be claimed to have made the major contribution to the end result.

Aspects of the notion of emergent knowledge have been studied in animal behavior and are readily discernible in human beings. Maturana (1960) found on the basis of experiments that when a frog catches a fly, not just specific components, but the greater part of the nervous system of the frog is involved in setting the frog into a 'fly catching' state. The components appear to interact in some manner to cooperatively contribute to the final act of catching the fly. Similarly, when human beings that have been successful in exercising a very high degree of skill in some particular endeavor (e.g., tight-rope walking, athletic competition, or a game of chess) are asked to explain the underlying reasons for their capabilities they will inevitably point to the importance of general mental preparation for the event rather than any specific practice exercise.

3rd Wave software will attempt to emulate the learning capabilities of the infant and child rather than try to model the existing world in the computer. It is true that 2nd Wave software will always be limited in its ability to deal with unforeseen conditions. The best currently available industrial robot may perform superbly in a modern factory for which it was designed, but will fail miserably in the relatively unstructured environment of an older less mechanized factory. The challenge for 3rd Wave software is to build agents that learn through their interaction with the natural environment (Figure 14). These agents must incorporate a much more sophisticated set of communication capabilities than are currently available or foreseen in the near future for either service-agents or object-agents in 2nd Wave software. Such capabilities include artificial vision, movement, touch, and hearing. While this is a formidable research agenda in itself, the development of the low level internal software components that are necessary to cooperatively process the environmental stimuli into knowledge and understanding presents an even more daunting task.

While progress is being made in the development of robotic agents, with some astonishing successes, it is unlikely that 3rd Wave software with significant learning capabilities (i.e., learning capabilities that are comparable to human capabilities, even though they might be notably inferior) will become available during the next two decades. In the meantime,

multi-agent, knowledge-based systems will greatly advance human decision-making capabilities and at the same time provide a necessary test bed for gaining experience with human-computer partnership environments. The value and need for this experience must not be underestimated. It is a prerequisite for the effective application of the rapid technological advances that are expected to continue in the information systems field.

2.4 A Question of 'Intelligence'

Surely, computer intelligence is a misnomer? From a commonsense point of view it would appear that humans have intelligence and computers are just very fast but unintelligent machines. Looking at this question from an entirely human point of view we may well come to such a conclusion. However, are there different kinds of intelligence? In other words, is intelligence something that is entirely reserved for living beings or can a machine display behavior that is akin to intelligence?

Before attempting to answer this question we should perhaps first address another seemingly less difficult question: Are there different levels of intelligence? If there are levels of intelligence then remembering is probably the lowest level of intelligence. Certainly computers can store vast amounts of data and can retrieve this data quickly and accurately. The immediate response might be that remembering is more than just retrieving data. Remembering also involves relationships and context. It is this context that makes data meaningful and relevant.

Interestingly enough that is what the current paradigm shift in computer software design and development is all about. We are moving from a data-centric to an information-centric software environment. What this really means is that we are representing information rather than data in the computer (i.e., information is data with relationships to provide some degree of context). This allows us to include modules in the software that are able to automatically reason (more recently referred to as agents) and communicate the results of their reasoning activities to other agents (including human users). One could argue that in some respects we are able to create in this way a virtual copy of a problem situation, or even a limited form of human society, in the computer environment. The players (i.e., the agents) in this virtual society can assume many different roles and can contribute and collaborate at many levels; - from the most primitive to the more sophisticated levels.

So, it seems that if we are careful to store not only data in the computer but also the relationships that convert such numbers and words into information then we can also embed in the software rule sequences that are capable of reasoning about this information. Such sequences may be as simple as condition-action statements. For example, ***if*** an enemy tank unit is sighted ***then*** place a call-for-fire on the enemy tank unit ***and*** commence the process of weapon selection. In this way we can implement, through the use of computer software, at least some elementary automatic reasoning capabilities in computers. As an example, a decision-support application for expeditionary warfare involving sea-basing operations might include agents that perform very elementary tasks such as calculating the fuel consumed by a helicopter in transporting supplies from the sea base to an inland supply point.

However, the same application might also incorporate agents that perform more sophisticated tasks. For example, selecting the best mix of lift assets (e.g., helicopters, hovercraft, vertical take-off aircraft, etc.) to transport a wide range of supplies to multiple landing zones within requested time windows, and within constraints such as weather conditions, enemy actions, and so on. The latter agents consider results received from other agents, and utilize a wide range of heuristic and algorithmic methods to arrive at a possible solution. In some respects this is similar to human society where problems are often solved through a team effort. In such teams some people contribute very simple capabilities and others contribute more sophisticated capabilities.

However, in respect to the sea-basing software application discussed above one might ask: Are the combined actions of these agents totally predictable? The answer is, no. While the results produced by the simple agents are certainly predictable, the impact that these results may have on the collective actions of the system is not necessarily predictable. In other words, the intelligence of the software system derives from the interactions (or more appropriately the collaboration) of the communicating elements of the system (i.e., the agents). With some exceptions one would generally not attribute intelligence to any single agent in current command and control information-centric software systems. However, such systems do display a collective intelligence that is not necessarily predictable and that can be quite powerful.

It therefore seems that we human beings must be willing to accept the proposition that there are different types of intelligence. In other words, intelligence cannot be measured only in human terms. There is no a priori reason to assume that computer intelligence is like human intelligence. In a corollary sense, it is unlikely to be productive to attempt to create a single software agent with human-like intelligence. A better approach is to look upon software as a virtual environment in which many software agents (utilizing their automatic reasoning capabilities, both reactively and proactively) navigate themselves into a solution area, through their countless interactions (i.e., collaborations). When we add the human user (i.e., human intelligence) to this environment we increase the potential capabilities of the system manifold.

3. Decision Making as a Human Cognitive Activity

The purpose of this Section is to present some understandings of the human problem solving activity that we have gained in the Collaborative Agent Design Research Center (CADRC) over the past decade. Since we feel strongly that the human decision maker should be an integral component of any computer-based decision-support system, it follows that we would have endeavored to incorporate many of the elements that appear to be important to the user in the design of these system. The complexity of the human cognitive system is evidenced by the large body of literature that describes problem solving behavior and the relatively fewer writings that attempt to provide comprehensive explanations of this behavior. Our contributions in this field are confined to the identification of important elements of the problem solving activity and exploration of how these elements will influence the design of a decision-support system.

3.1 Some Human Problem Solving Characteristics

Human beings are inquisitive creatures who seek explanations for all that they observe and experience in their living environment. While this quest for understanding is central to our success in adapting to a changing and at times unforgiving environment, it is also a major cause for our willingness to accept partial understandings and superficial explanations when the degree of complexity of the problem situation confounds our mental capabilities. In other words, a superficial or partial explanation is considered better than no explanation at all. As flawed as this approach may be, it has helped us to solve difficult problems in stages. By first oversimplifying a problem we are able to develop an initial solution that is later refined as a better understanding of the nature of the problem evolves. Unfortunately, now we have to contend with another characteristic of human beings, our inherent resistance to change and aversion to risk taking. Once we have found an apparently reasonable and workable explanation or solution we tend to lose interest in pursuing its intrinsic shortcomings and increasingly believe in its validity. Whether driven by complacency or lack of confidence, this state of affairs leads to many surprises. We are continuously discovering that what we believed to be true is only partly true or not true at all, because the problem is more complicated than we had previously assumed.

At times a particular set of explanations, or school of thought, becomes entrenched as a paradigm that is not easily broken. Kuhn (1977) has drawn attention to the stagnating influence on progress of scientific paradigms, the resistance experienced by individuals or small groups that wish to correct flaws in a paradigm, and the resurgence of innovative activity after the paradigm has been broken. If experts in science will succumb to this weakness in human nature then how much more difficult will it be for a layperson to maintain a discerning mind? This could well become a serious consideration in problem areas where technology advances at a rapid rate, and these advances are almost immediately translated into commercial products that are available to large sections of the community. One such example is, of course, the rapid development of computers and communication systems and their potential assistance in human decision-making endeavors.

The complexity of problems faced by human society in areas such as management, economics, marketing, engineering design, and environmental preservation, is increasing for several reasons. First, computer-driven information systems have expanded these areas from a local to an increasingly global focus. Even small manufacturers are no longer confined to a regionally localized market for selling their products. The marketing decisions that they have to make must take into account a wide range of factors and a great deal of knowledge that is far removed from the local environment. Second, as the scope of the problem system increases so do the relationships among the various factors. These relationships are difficult to deal with, because they require the decision-maker to consider many factors concurrently. Although the biological operation of the human brain is massively parallel, our conscious reasoning processes are sequential. Simply stated, we have difficulty reasoning about more than two or three variables at any one time. Third, as the scope of problems increases decision-makers suffer simultaneously from two diametrically opposed but related conditions. They tend to be overwhelmed by the shear volume of information that they have to consider, and yet they lack information in many specific areas. To make matters worse, the information tends to change dynamically in largely unpredictable ways

It is therefore not surprising that governments, corporations, businesses, down to the individual person, are increasingly looking to computer-based decision-support systems for assistance. This has placed a great deal of pressure on software developers to rapidly produce applications that will overcome the apparent failings of the human decision-maker. While the expectations have been very high, the delivery has been much more modest. The expectations were simply unrealistic. It was assumed that advances in technology would be simultaneously accompanied by an understanding of how these advances should be applied optimally to assist human endeavors. History suggests that such an a priori assumption is not justified. There are countless examples that would suggest the contrary. For example, the invention of new materials (e.g., plastics) has inevitably been followed by a period of misuse. Whether based on a misunderstanding or lack of knowledge of its intrinsic properties, the new material was typically initially applied in a manner that emulated the material(s) it replaced. In other words, it took some time for the users of the new material to break away from the existing paradigm. A similar situation currently exists in the area of computer-based decision-support systems.

3.1.1 The Rationalistic Tradition

To understand current trends in the evolution of progressively more sophisticated decision-support systems it is important to briefly review the foundations of problem solving methodology from an historical perspective. Epistemology is the study or theory of the origin, nature, methods and limits of knowledge. The dominant epistemology of Western Society has been technical rationalism (i.e., the systematic application of scientific principles to the definition and solution of problems).

The rationalistic approach to a problem situation is to proceed in well defined and largely sequential steps (Figure 15): define the problem; establish general rules that describe the relationships that exist in the problem system; apply the rules to develop a solution; test

the validity of the solution; and, repeat all steps until an acceptable solution has been found. This simple view of problem solving suggested a model of sequential decision making that has retained a dominant position to the present day. With the advent of computers it was readily embraced by 1st Wave software because of the ease with which it could be translated into decision-support systems utilizing the procedural computer languages that were available at the time.

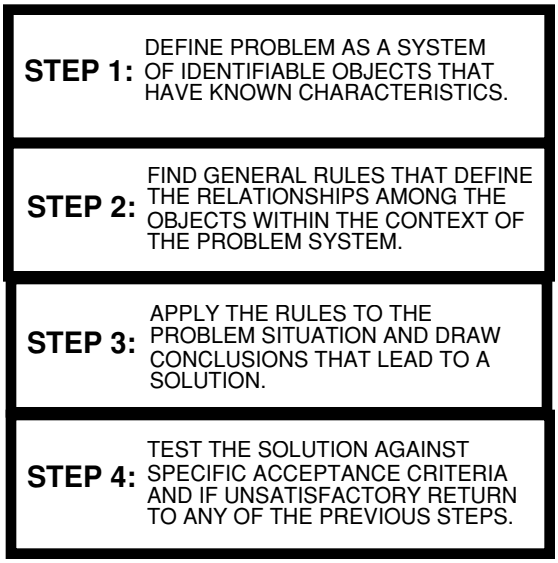


Figure 15: Solution of Simple Problems

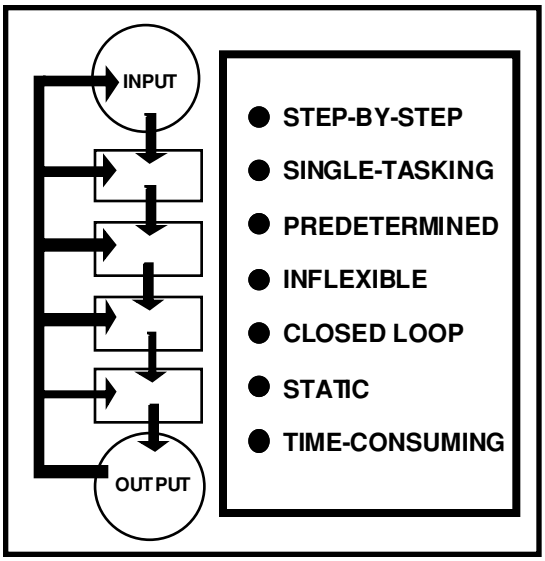


Figure 16: Sequential Decision Support

The close correlation between the rationalistic approach and what is commonly referred to as the scientific method, is readily apparent in the series of basic steps that are employed in scientific investigations: observe the phenomenon that requires explanation; formulate a possible explanation; develop a method capable of predicting or generating the observed phenomenon; interpret the results produced by the method; and, repeat all steps until an acceptable explanation of the observed phenomenon has been found. Scientific research typically attempts to establish situations in which observable actions (or reactions) are governed by a small number of variables that can be systematically manipulated. Every effort is made to keep the contrived situation simple, clear and deterministic, so that the results of the simulation can be verified.

However, natural phenomena and real world problems are often very complex involving many related variables. Neither the relationships among the variables nor the variables themselves are normally sufficiently well understood to provide the basis for clear and comprehensive definitions. In other words, problem situations are often too complex to be amenable to an entirely logical and predefined solution approach. Under these conditions the analytical strategy has been to decompose the whole into component parts, as follows:

- Decompose the problem system into sub-problems.

- Study each sub-problem in relative isolation, using the rationalistic approach (Figure 15). If the relationships within the sub-problem domain cannot be clearly defined then decompose the sub-problem further.

- Combine the solutions of the sub-problems into a solution of the whole.

Underlying this problem solving strategy is the implicit assumption that an understanding of parts leads to an understanding of the whole. Under certain conditions this assumption may be valid. However, in many complex problem situations the parts are tightly coupled so that the behavior of the whole depends on the interactions among the parts rather than the internal characteristics of the parts themselves (Bohm 1983, Senge 1993). An analogy can be drawn with the behavior of ants. Each ant has only primitive skills, such as the ability to interpret the scent of another ant and the instinctive drive to search for food, but little if any notion of the purpose or objectives of the ant colony as a whole. In other words, an understanding of the behavior of an individual ant does not necessarily lead to an understanding of the community behavior of the ant colony of which the ant is a part.

Decomposition is a natural extension of the scientific approach to problem solving and has become an integral and essential component of rationalistic methodologies. Nevertheless, it has serious limitations. First, the behavior of the whole usually depends more on the interactions of its parts and less on the intrinsic behavior of each part. Second, the whole is typically a part of a greater whole and to understand the former we have to also understand how it interacts with the greater whole. Third, the definition of what constitutes a part is subject to viewpoint and purpose, and not intrinsic in the nature of the whole. For example, from one perspective a coffee maker may be considered to comprise a bowl, a hotplate, and a percolator. From another perspective it consists of electrical and constructional components, and so on.

Rationalism and decomposition are certainly useful decision-making tools in complex problem situations. However, care must be taken in their application. At the outset it must be recognized that the reflective sense (Schon 1983) and intuition of the decision-maker are at least equally important tools. Second, decomposition must be practiced with restraint so that the complexity of the interactions among parts is not overshadowed by the much simpler behavior of each of the individual parts. Third, it must be understood that the definition of the parts is largely dependent on the objectives and knowledge about the problem that is currently available to the decision-maker. Even relatively minor discoveries about the greater whole, of which the given problem situation forms a part, are likely to have significant impact on the purpose and the objectives of the problem situation itself.

3.1.2 Decision Making in Complex Problem Situations

At the beginning of this Technical Report (see Section 1) we stressed the importance of internal and external relationships in complex problem situations. As shown in Figure 17, there are several characteristics that distinguish a complex problem from a simple problem. First, the problem is likely to involve many related issues or variables. As

discussed earlier the relationships among the variables often have more bearing on the problem situation than the variables themselves. Under such tightly coupled conditions it is often not particularly helpful, and may even be misleading, to consider issues in isolation. Second, to confound matters some of the variables may be only partially defined and some may yet to be discovered. In any case, not all of the information that is required for formulating and evaluating alternatives is available. Decisions have to be made on the basis of incomplete information.

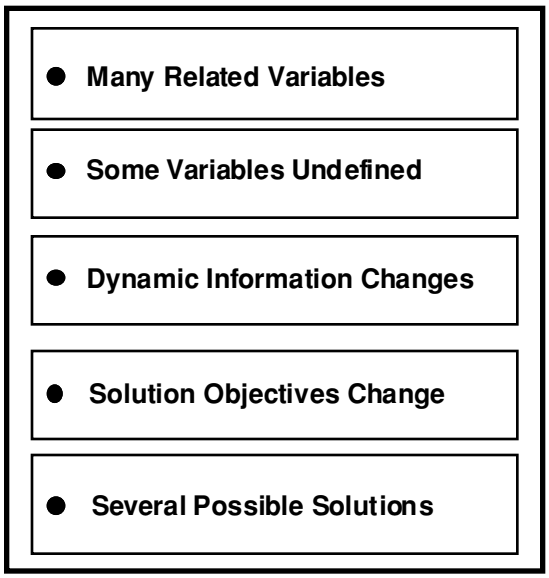


Figure 17: Character of Complex Problems

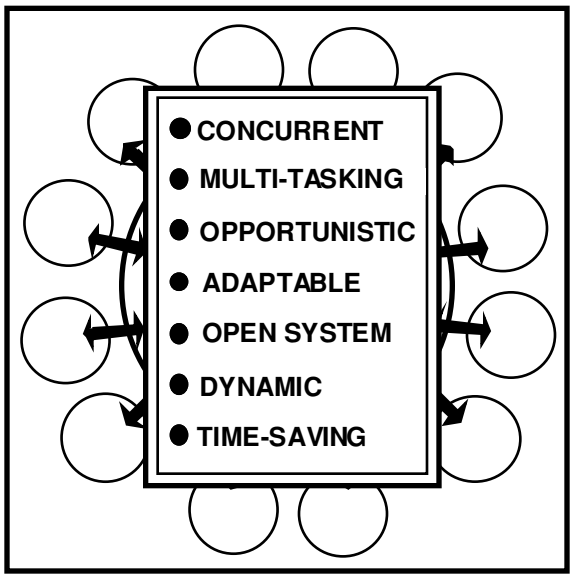


Figure 18: Parallel Decision Support

Third, complex problem situations are pervaded with dynamic information changes. These changes are related not only to the nature of an individual issue, but also to the context of the problem situation. For example, a change in location of an enemy force (even within the same sector of the battlefield) could easily have a major impact on the entire nature of the combat situation facing the commander. Apart from the disposition of friendly forces under these changed conditions, the influence on target priorities, and the effectiveness of available weapon assets, such a relocation could call into question the very feasibility of the existing course of action (i.e., the battle plan). Even under less critical conditions it is not uncommon for the solution objectives to change several times during the decision-making process. This fourth characteristic of complex problem situations is of particular interest. It exemplifies the tight coupling that can exist among certain problem issues, and the degree to which decision-makers must be willing to accommodate fundamental changes in the information that drives the problem situation.

Fifth, complex problems typically have more than one solution (Archea 1987). It is usually fruitless to look for an optimum solution, because there are no static benchmarks available for evaluating optimality. A solution is found to be acceptable if it satisfies certain performance requirements and if it has been determined that the search for alternatives is no longer warranted. Such a determination is often the result of resource

constraints (e.g., availability of time, penalty of non-action, or financial resources) rather than a high level of satisfaction with the quality of the proposed solution.

While human decision-making in complex problem situations has so far defied rigorous scientific explanation, we do have knowledge of at least some of the characteristics of the decision making activity.

- Decision-makers typically define the problem situation in terms of issues that are known to impact the desired outcome. The relative importance of these issues and their relationships to each other change dynamically during the decision-making process. So also do the boundaries of the problem space and the goals and objectives of the desired outcome. In other words, under these circumstances decision-making is an altogether dynamic process in which both the rules that govern the process and the required properties of the end result are subject to continuous review, refinement and amendment.
- The complexity of the decision-making activity does not appear to be due to a high level of difficulty in any one area but the multiple relationships that exist among the many issues that impact the desired outcome. Since a decision in one area will tend to influence several other areas there is a need to consider many factors at the same time. This places a severe burden on the human cognitive system. Although the neurological mechanisms that support conscious thought processes are massively parallel, the operation of these reasoning capabilities is largely sequential. Accordingly, decision-makers tend to apply simplification strategies for reducing the complexity of the problem solving activity. In this regard it becomes readily apparent why 2nd Wave software provides a much more useful architecture for decision-support systems (Figure 18).
- Observation of decision-makers in action has drawn attention to the important role played by experience gained in past similar situations, knowledge acquired in the general course of decision-making practice, and expertise contributed by persons who have detailed specialist knowledge in particular problem areas. The dominant emphasis on experience is confirmation of another fundamental aspect of the decision-making activity. Problem-solvers seldom start from first principles. In most cases, the decision-maker builds on existing solutions from previous situations that are in some way related to the problem under consideration. From this viewpoint, the decision-making activity involves the modification, refinement, enhancement and combination of existing solutions into a new hybrid solution that satisfies the requirements of the given problem system. In other words, problem solving can be described as a process in which relevant elements of past prototype solution models are progressively and collectively molded into a new solution model. Very seldom are new prototype solutions created that do not lean heavily on past prototypes.

- Finally, there is a distinctly irrational aspect to decision-making in complex problem situations. Donald Schon refers to a "...reflective conversation with the situation...". (Schon 1983). He argues that decision-makers frequently make value judgments for which they cannot rationally account. Yet, these intuitive judgments often result in conclusions that lead to superior solutions. It would appear that such intuitive capabilities are based on a conceptual understanding of the situation, which allows the problem solver to make knowledge associations at a highly abstract level.

Based on these characteristics the solution of complex problems can be categorized as an information intensive activity that depends for its success largely on the availability of information resources and, in particular, the experience and reasoning skills of the decision-makers. It follows that the quality of the solutions will vary significantly as a function of the problem-solving skills, knowledge, and information resources that can be brought to bear on the solution process. This clearly presents an opportunity for the useful employment of computer-based decision-support systems in which the capabilities of the human decision-maker are complemented with knowledge bases, expert agents, and self-activating conflict identification and monitoring capabilities.

3.2 Principal Elements of Decision Making

Over the past decade that the CADRC has been developing distributed, collaborative decision-support systems some insights have been gained into the nature of the decision-making activity. In particular, we have found it useful to characterize decision-making in terms of six functional elements (Figure 19): information; representation; visualization; communication; reasoning; and, intuition.

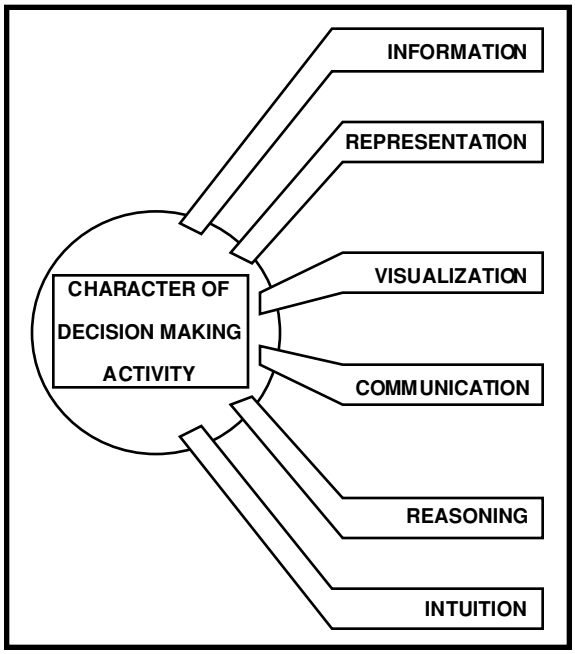


Figure 19: Principal Decision-Making Elements

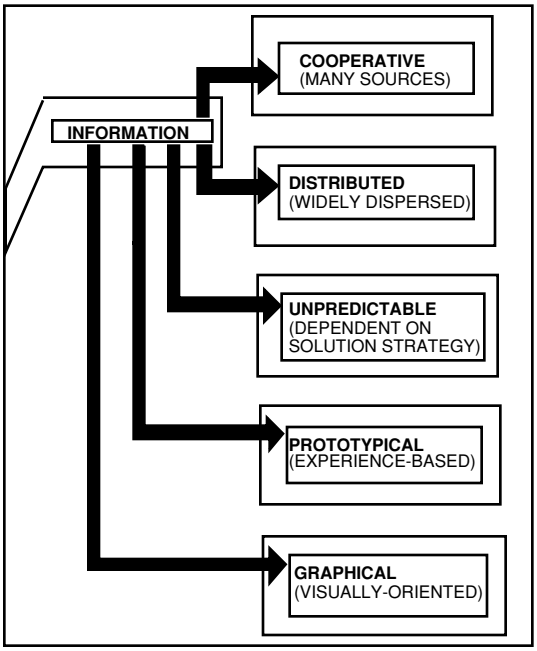


Figure 20: The 'Information' Element

3.2.1 The 'Information' Element

Decision-making in complex problem situations is a cooperative activity involving many sources of information that are often widely dispersed. Seldom is all of the information required for the solution, or even only a component of the problem, physically located in the immediate vicinity of the decision-maker. In fact, much of the information is likely to reside in remote depositories that can be accessed only through electronic means, the telephone, or the temporary relocation of a member of the problem-solving team (Figure 20). If the desired information requires expert advice the services of a consultant may be required in addition to, or instead of, access to an information resource.

The term 'information' is used here in the broadest sense to include not only factual data and the progressively more comprehensive and detailed description of the problem system, but also the many knowledge bases that are part of the local and global environment within which the problem situation is constituted. In this regard, we are concerned with the knowledge of the individual members of the problem-solving team, the knowledge of peripheral players (e.g., colleagues, associates and consultants), the collective knowledge of the profession (such as the various engineering professions, the military establishment, or the management profession) and industry, and beyond that those aspects of what might be referred to as 'global knowledge' that impacts the problem context.

Typically, the problem specifications (i.e., constraints, criteria, and objectives) evolve with the problem solution as the decision-makers interact with the problem situation. Accordingly, the information requirements of the problem solver are not predictable since the information needed to solve the problem depends largely on the solution strategy adopted (Fischer and Nakakoji 1991). In this respect problem solving is a learning process in which the decision-maker progressively develops a clearer understanding of the problem that is required to be solved. Much of the information that decision-makers use in the development of a problem solution is gleaned from experience with past projects. In fact, it can be argued that solutions commonly evolve out of the adaptation, refinement and combination of prototypes (Gero et al. 1988). This argument suggests that the more expert human decision-makers are the more they tend to rely on prototypical information in the solution of complex problems. It would appear that the accumulation, categorization and ability to apply prototype knowledge is the fundamental requirement for a human decision-maker to reach the level of 'expert' in a particular domain. Based largely on the work of Gero et al. (1988) and Rosenman and Gero (1993) the following techniques used by engineering designers to develop solutions through the manipulation of prototypes can be identified as being universally applicable to other problem domains:

- ***Refinement:*** The prototype can be applied after changes have been made in the values of parameter variables only (i.e., the instance of the prototype is reinterpreted within the acceptable range of the parameter variables).
- ***Adaptation:*** Application of the prototype requires changes in the parameters that constitute the description of the prototype instance, based

on factors that are internal to the prototype (i.e., a new prototype instance is produced).

- ***Combination:*** Application of the prototype requires the importation of parameter variables of other prototypes, producing a new instance of a reinterpreted version of the original prototype.

- ***Mutation:*** Application of the prototype requires structural changes to the parameter variables, either through internal manipulations or the importation of parameter variables from external sources (i.e., either a reinterpreted version of the original prototype or a new prototype is produced).

- ***Analogy:*** Creation of a new prototype based on a prototype that exists in another context, but displays behavioral properties that are analogous to the application context.

For application purposes in knowledge-based decision-support systems prototypes may be categorized into five main groups based on knowledge content (Schon 1988, Pohl and Myers 1994):

1. **Vertica**l prototype knowledge bases that contain typical object descriptions and relationships for a complete problem situation or component thereof. Such a knowledge base may include all of the types that exist in a particular problem setting, for example: an operational template for a particular kind of military mission; a certain type of propulsion unit; or, a building type such as a library, sports stadium, or supermarket.

2. **Horizontal** prototype knowledge bases that contain typical solutions for sub-problems such as logistical procurement practices, construction of a temporary bunker, or techniques for repairing equipment. This kind of knowledge often applies to more than one discipline. For example, the techniques for repairing a truck apply equally to the military as they do to auto-repair shops, engineering concerns, and transportation related organizations.

3. **Domain** prototype knowledge bases that contain guidelines for developing solutions within contributing narrow domains. For example, the range of structural solutions appropriate for the construction of a suspension bridge during a military mission is greatly influenced by the availability of material, the prevailing wind conditions, and the time available for erection. Posed with this design problem military engineers will immediately draw upon a set of rules that guide the design activity.

4. **Exemplar** prototype knowledge bases that describe a specific instance of an object type or solution to a sub-problem. Exemplary prototypes can be instances of vertical or horizontal prototypes, such as a particular fortification

(e.g., bunker) or a specific type of artillery mount. Decision- makers often refer to exemplary prototypes in exploring solution alternatives to sub-problems.

5. **Experiential** knowledge bases that represent the factual prescriptions, strategies and solution conventions employed by the decision-maker in solving similar kinds of problem situations. Such knowledge bases are typically rich in methods and procedures. For example, a particularly memorable experience such as the deciding event in a past business negotiation or the turning point of a military offensive, may provide the basis for a solution method that is applied later to create a similar experience in a new problem situation that may be quite different in most other respects. In other words, experiential prototypes are not bound to a specific type of problem situation. Instead, they represent techniques and methods that can be reproduced in various contexts with similar results. Experiential knowledge is often applied in very subtle ways to guide the solution of sub-problems (e.g., a subterfuge in a military maneuver that is designed to mislead the enemy).

The volume of prototypical information is potentially overwhelming. However, the more astute and experienced decision-maker will insist on taking time to assimilate as much information as possible into the problem setting before committing to a solution theme. There is a fear that early committal to a particular solution concept might overlook characteristics of the problem situation that could gain in importance in later stages, when the solution has become too rigid to adapt to desirable changes. This reluctance to come to closure places a major information management burden on the problem solver. Much of the information cannot be specifically structured and prepared for ready access, because the needs of the problem solver cannot be fully anticipated. Every step toward a solution generates new problems and information needs (Simon 1981).

3.2.2 The 'Representation' Element

The methods and procedures that decision-makers utilize to solve complex problems rely heavily on their ability to identify, understand and manipulate objects (Figure 21). In this respect, objects are complex symbols that convey meaning by virtue of the explicit and implicit information that they encapsulate within their domain. For example, military strategists develop operational plans by reasoning about terrain, weather conditions, enemy positions, weapon assets, and so on. Each of these objects encapsulates knowledge about its own nature, its relationships with other objects, its behavior within a given environment, what it requires to meet its own performance objectives, and how it might be manipulated by the decision-maker within a given problem scenario (Figure 22). This knowledge is contained in the various representational forms of the object as factual data, relationships, algorithms, rules, exemplar solutions, and prototypes.

The reliance on object representations in reasoning endeavors is deeply rooted in the innately associative nature of the human cognitive system. Information is stored in long-term memory through an indexing system that relies heavily on the forging of association

paths. These paths relate not only information that collectively describes the meaning of symbols such as 'helicopter', 'rifle' and 'truck', but also connect one symbol to another. The symbols themselves are not restricted to the representation of physical objects, but also serve as concept builders. They provide a means for grouping and associating large bodies of information under a single conceptual metaphor. In fact, Lakoff and Johnson (1980) argue that "...our ordinary conceptual system, in terms of which we both think and act, is fundamentally metaphorical in nature...". They refer to the influence of various types of metaphorical concepts, such as 'desirable is up' (spatial metaphors) and 'fight inflation' (ontological or human experience metaphors), as the way human beings select and communicate strategies for dealing with every day events.

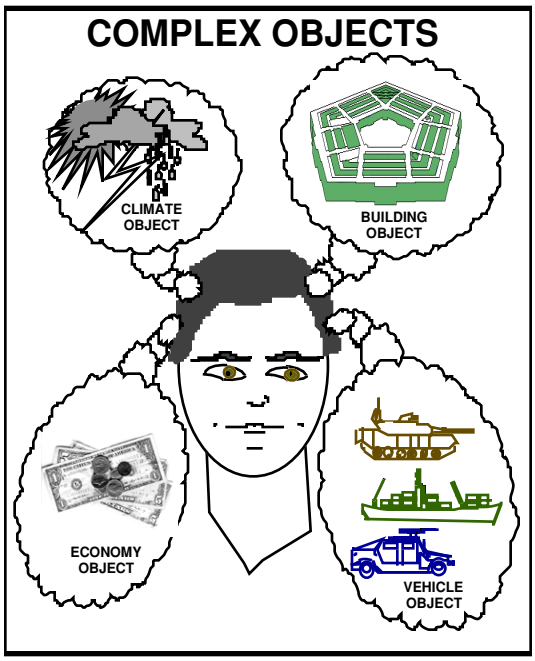


Figure 21: Symbolic Reasoning with Objects

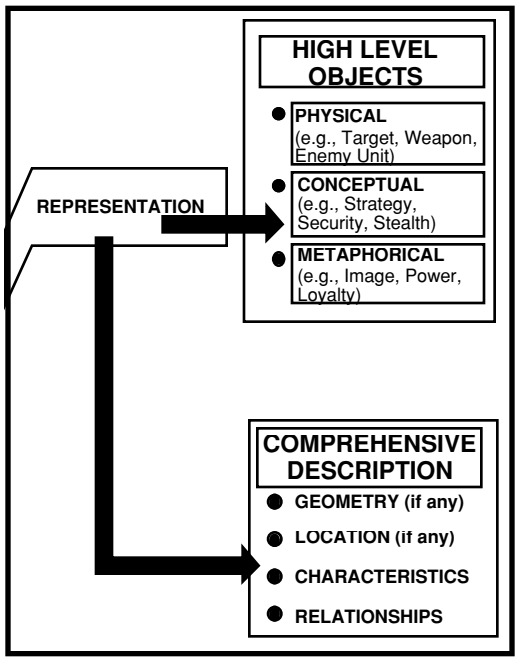


Figure 22: The 'Representation' Element

Problem solvers typically intertwine the factually based aspects of objects with the less precise, but implicitly richer language of metaphorical concepts. This leads to the spontaneous linkage of essentially different objects through the process of analogy. In other words, the decision-maker recognizes similarities between two or more sub-components of apparently unrelated objects and embarks upon an exploration of the discovered object seeking analogies where they may or may not exist. At times these seemingly frivolous pursuits lead to surprising and useful solutions of the problem at hand.

Referring again to our previous discussions of this topic (see Sections 1.4.5 and 2.3.2) the need for a high level representation is fundamental to all computer-based decision-support systems. It is an essential prerequisite for embedding artificial intelligence in such systems, and forms the basis of any meaningful communication between user and computer. Without a high level representation facility the abilities of the computer to assist the human decision maker are confined to the performance of menial tasks, such as

the automatic retrieval and storage of data or the computation of mathematically defined quantities. While even those tasks may be highly productive they cannot support a partnership in which human users and computer-based systems collaborate in a meaningful and intelligent manner in the solution of complex problems.

3.2.3 The 'Visualization' Element

Problem solvers use various visualization media, such as visual imagination, drawings and physical models, to communicate the current state of the evolving solution to themselves and to others (Figure 23). Drawings, in particular, have become intrinsically associated with problem solving. Although the decision-maker can reason about complex problems solely through mental processes, drawings and related physical images are useful and convenient for extending those processes. The failings of the drawing as a vehicle for communicating the full intent of the decision-maker do not apply to the creator of the drawing. To the latter the drawing serves not only as an extension of long term memory, but also as a visual bridge to its associative indexing structure. In this way, every meaningful part of the drawing is linked to related data and deliberation sequences that together provide an effectively integrated and comprehensive representation of the artifact.

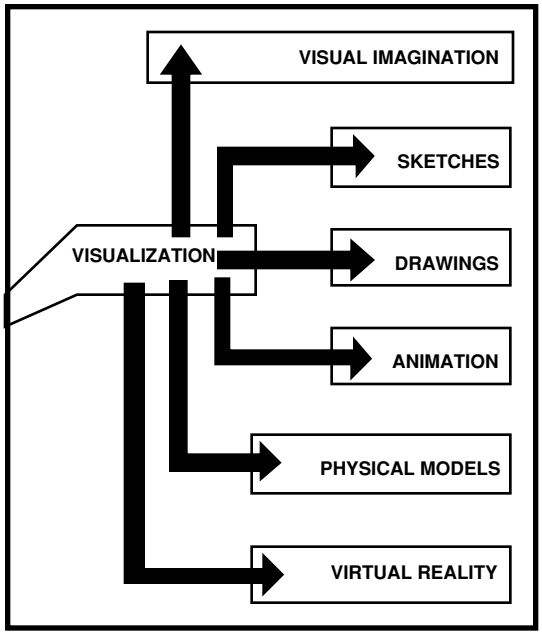


Figure 23: The 'Visualization' Element

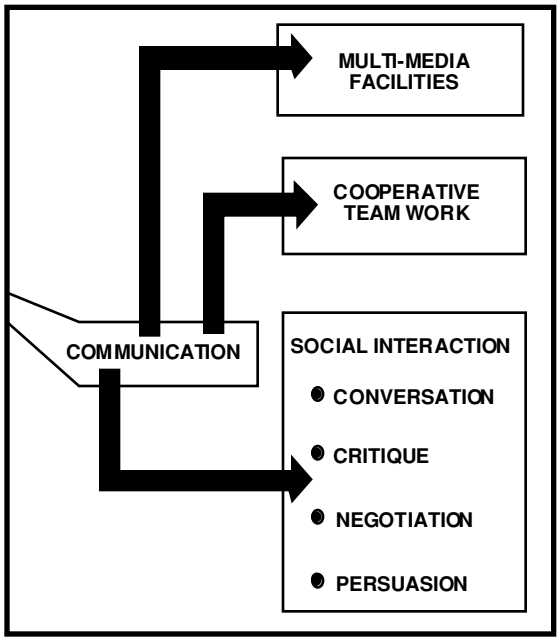


Figure 24: The 'Communication' Element

From a technical point of view a great deal of headway has been made over the past two decades in the area of computer-based visualization. However, without high level representation capabilities even the most sophisticated computer generated images are nothing but hollow shells. If the computer system does not have even the simplest understanding of the nature of the objects that are contained in the image then it cannot contribute in any way to the analysis of those objects. On the other hand, visualization in combination with high level representation becomes the most powerful element of the

user-interface of a decision-support system. Under these circumstances, visualization promotes the required level of understanding between the user and the computer as they collaborate in the solution of the problem.

3.2.4 The 'Communication' Element

The solution of complex problems is typically undertaken by a team of decision-makers. Each team member contributes within a cooperative decision-making environment that relies heavily on the normal modes of social interaction, such as conversation, critique, negotiation, and persuasion (Figure 24). Two aspects of such an interactive environment are particularly well catered for in computer-based systems. The first aspect relates to the ability of computer-driven communication networks to link together electronically-based resources located anywhere in the world or space. Technical advances in the communication industry have greatly enhanced the ability of individuals to gain access to remotely distributed information sources, and to interact with each other over vast distances. In fact, connectivity rather than geographical distance has become the principal determinant of communication.

The second aspect is interwoven with the first by recent technological advances that permit all types of information to be converted into digital form. Through the use of digital switching facilities modern communication networks are able to transmit telephone conversations and graphical images in the same way as data streams have been sent from one computer to another over the past 30 years.

As a direct result of these advances in communication systems the convenient and timely interaction of all of the members of a widely dispersed problem-solving team is technically assured. It is now incumbent on software developers to produce computer-based design systems that can fully support cooperative teamwork that is neither geographically nor operationally limited. Such systems will integrate not only computer-based information resources and agents, but also multiple human agents (i.e., users) who will collaborate with the computer-based resources in a real-time interactive environment. While the basic technology for this level of communication is already in place further advances are expected in the area of transmission speed and the computer system software that will facilitate message passing within heterogeneous networks in a user-transparent fashion.

3.2.5 The 'Reasoning' Element

Reasoning is central to any decision-making activity. It is the ability to draw deductions and inferences from information within a problem-solving context. The ability of the problem solver to reason effectively depends as much on the availability of information, as it does on an appropriately high level form of object representation (Figure 25). Decision-makers typically define complex problems in terms of issues that are known to impact the desired outcome. The relative importance of these issues and their relationships to each other change dynamically during the decision-making process. So

also do the boundaries of the problem space and the goals and objectives of the desired outcome. In other words, the solution of complex problems is an altogether dynamic process in which both the rules that govern the process and the required properties of the end result are subject to continuous review, refinement and amendment (Reitman 1964 and 1965, Rittel and Weber 1984).

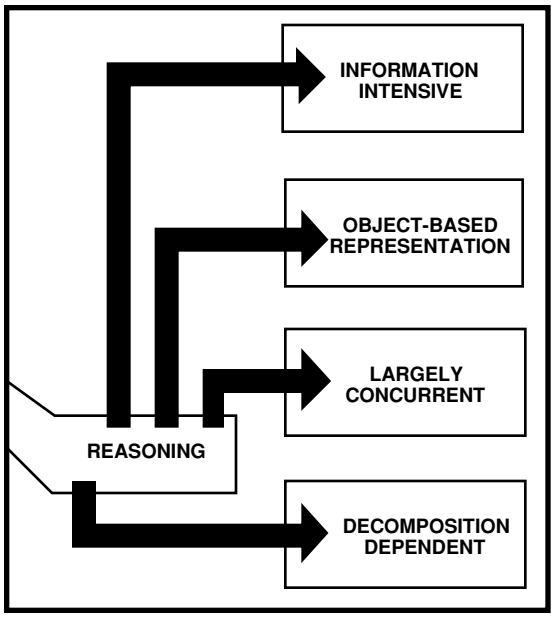


Figure 25: The 'Reasoning' Element

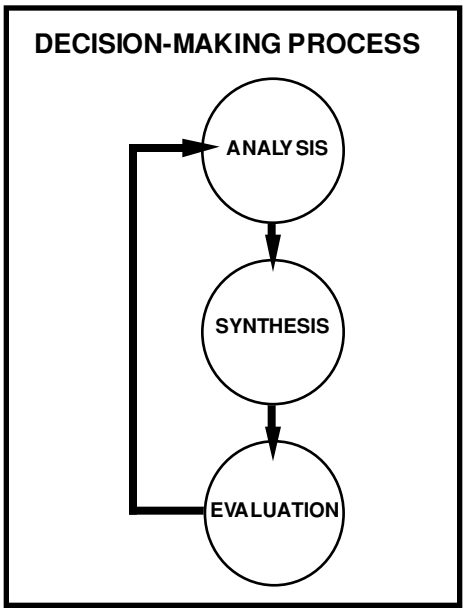


Figure 26: Reasoning Methodology

As discussed previously, the complexity of a problem is normally not due to a high degree of difficulty in any one area but the multiple relationships that exist among the many issues that impact the desired outcome. Since a decision in one area will tend to influence several other areas there is a critical need for concurrency. However, the reasoning capabilities of the human problem solver are sequential in nature. Accordingly, decision-makers find it exceedingly difficult to consider more than three or four issues at any one time. In an attempt to deal with the concurrency requirement several strategies are commonly employed to reduce the complexity of the reasoning process to a manageable level.

- ***Constraint Identification:*** By sifting through the available information the problem solver hopes to find overriding restrictions and limitations that will eliminate knowledge areas from immediate consideration.

- ***Decision Factor Weighting:*** By comparing and evaluating important problem issues in logical groupings, relative to a set of predetermined solution objectives, the decision maker hopes to identify a smaller number of issues or factors that have greater impact on the final solution. Again, the strategy is to reduce the size of the information base by early elimination of apparently less important considerations.

- ***Solution Conceptualization:*** By adopting early in the decision-making process a conceptual solution, the problem solver is able to pursue a selective evaluation of the available information. Typically, the problem solver proceeds to subdivide the decision factors into two groups; those that are compatible with the conceptual solution and those that are in conflict. By a process of trial and error, often at a superficial level, the problem solver develops, adapts, modifies, re-conceives, rejects and, often, forces the preconceived concept into a final solution.

In complex problem situations reasoning proceeds in an iterative fashion through a cycle of analysis, synthesis and evaluation (Figure 26). During the analysis stage (Figure 27) the problem solver interprets and categorizes information to establish the relative importance of issues and to identify compatibilities and incompatibilities among the factors that drive these issues.

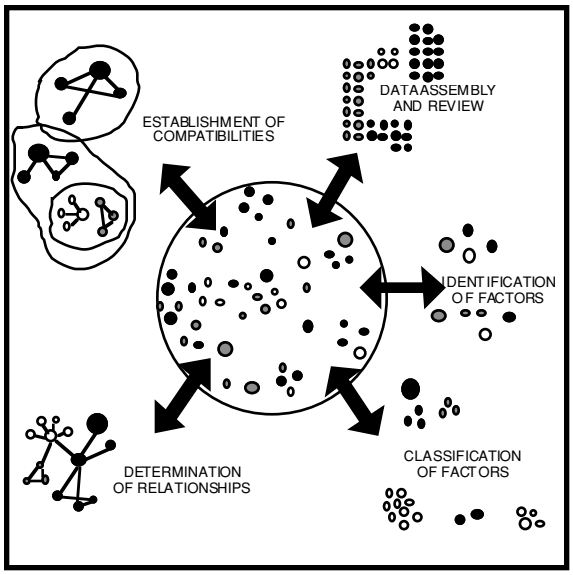


Figure 27: Analysis Stage of Reasoning

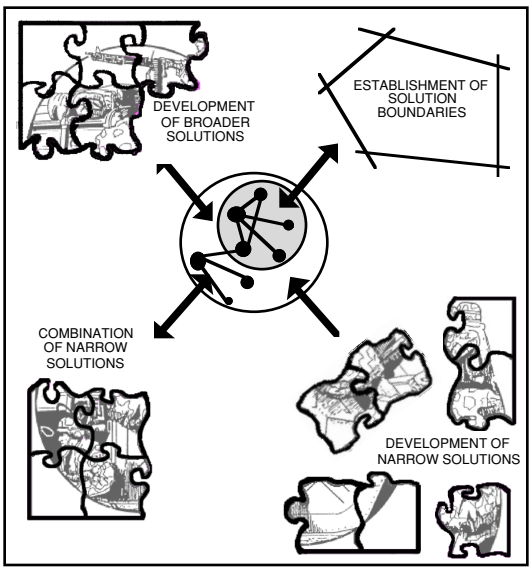


Figure 28: Synthesis Stage of Reasoning

During synthesis (Figure 28) solution boundaries and objectives are continuously reexamined as the decision-maker develops narrow solutions to sub-problems and combines these narrow solutions into broader solutions. Initially, these solution attempts are nothing more than trial balloons; - explorations based on the development of the relationships among the principal issues and compatible factors identified during the analysis stage. Later, as the problem-solving activity progresses, firmer conceptual solution strategies with broader implications emerge. However, even during later cycles these solution strategies tend to be based on a limited number of issues or factors.

During the evaluation stage (Figure 29) the decision-makers are forced to test the current solution strategy with all of the known problem issues, some of which may have been considered only superficially or not at all during the formulation of the current solution proposal. This may require the current solution concepts to be modified, extended or altogether replaced. Typically, several solution strategies are possible and none are

completely satisfactory. Archea (1987), in his description of the architectural design activity, refers to this activity as "...puzzle-making...", suggesting by implication that the decision maker utilizes the reasoning cycle more as a method for exploring the problem space than as a decision making tool for forcing an early solution.

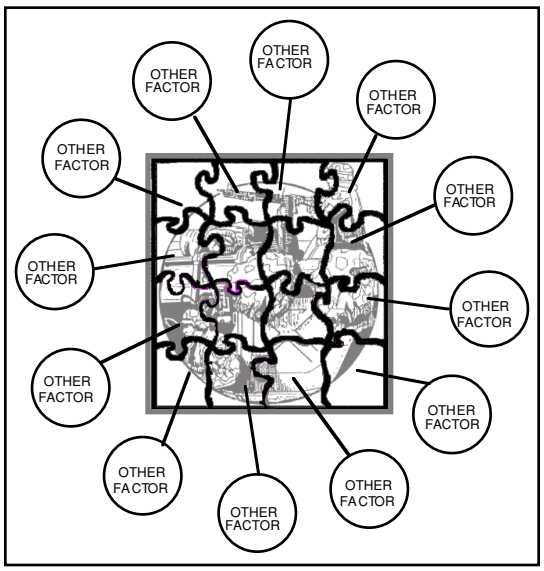


Figure 29: Evaluation Stage of Reasoning

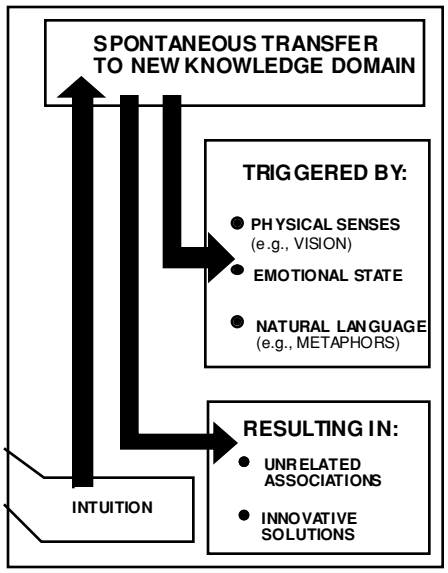


Figure 30: The 'Intuition' Element

3.2.6 The 'Intuition' Element

Donald Schon (1983 and 1988) has written extensively about the intuitive aspects of decision-making. Although he focused primarily on engineering design as an application area, his views provide valuable insight into the solution of complex problems in general. Design has all of the common characteristics of complex problem situations, and some additional ones such as the desire for solution uniqueness, that make it a prime candidate for computer-based assistance (Pohl et al.1994).

In Schon's (1988) view designers enter into "...design worlds..." in which they find the objects, rules and prototype knowledge that they apply to the design problem under consideration. The implication is that the designer continuously moves in and out of design worlds that are triggered by internal and external stimuli. While the reasoning process employed by the designer in any particular design world is typically sequential and explicitly logical, the transitions from state to state are governed by deeper physiological and psychological causes. Some of these causes can be explained in terms of associations that the designer perceives between an aspect or element of the current state of the design solution and prototype knowledge that the designer has accumulated through experience. Others may be related to emotional states or environmental stimuli, or interactions of both (Figure 30).

For example, applying Schon's view to the broader area of complex problem solving, a particular aspect of a problem situation may lead to associations in the decision-maker's

mind that are logically unrelated to the problem under consideration. However, when the decision-maker pursues and further develops these associations they sometimes lead to unexpected solutions. Typically, the validity of these solutions becomes apparent only after the fact and not while they are being developed. In popular terms we often refer to these solutions as 'creative leaps' and label the author as a brilliant strategist. What we easily forget is that many of these intuitions remain unrelated associations and do not lead to any worthwhile result. Nevertheless, the intuitive aspect of decision making is most important. Even if only a very small percentage of these intuitive associations were to lead to a useful solution, they would still constitute one of the most highly valued decision-making resources.

The reasons for this are twofold. First, the time at which the decision-maker is most willing to entertain intuitive associations normally coincides with a most difficult stage in the problem solving process. Typically, it occurs when an impasse has been reached and no acceptable solution strategy can be found. Under these conditions intuition may be the only remaining course of action open to the decision-maker. The second reason is particularly relevant if there is a strong competitive element present in the problem situation. For example, in command and control situations during the execution of military operations. Under these circumstances, strategies and solutions triggered by intuitive associations will inevitably introduce an element of surprise that is likely to disadvantage the adversary.

The importance of the 'intuition' element itself in decision-making would be sufficient reason to insist on the inclusion of the human decision-maker as an active participant in any computer-based decision system. In designing and developing such systems in the CADRC over the past decade we have come to appreciate the importance of the human-computer partnership concept, as opposed to automation. Whereas in some of our early systems (e.g., ICADS (Pohl et al. 1988) and AEDOT (Pohl et al. 1992)) we included agents that automatically resolve conflicts, today we are increasingly moving away from automatic conflict resolution to conflict detection and explanation. We believe that even apparently mundane conflict situations should be brought to the attention of the human agent. Although the latter may do nothing more than agree with the solution proposed by the computer-based agents, he or she has the opportunity to bring other knowledge to bear on the situation and thereby influence the final determination.

4. ICDM: An Application Development Framework

Over the past several years the Collaborative Agent Design Research Center (CADRC) at Cal Poly has developed a framework for implementing distributed, collaborative, multi-agent applications. We refer to this framework as the Integrated Cooperative Decision Making (ICDM) model. From a conceptually viewpoint, ICDM comprises an object-serving communication and coordination facility that integrates multiple computer-based agents and human agents (i.e., users) within a distributed knowledge-based environment (Figure 31). It consists of three principal components, shown schematically in Figure 32, as: a distributed database management system (DBMS) that has progressively evolved to provide at least some of the capabilities described in Section 4.2; a decision making advisory facility that coordinates various types of computer-based agents; and, multi-media user-interfaces through which human agents can interact with all components of the system and orchestrate the decision-making activities toward an acceptable solution.

4.1 The ICDM Model

The ICDM model is a collection of tools, application development experiences, and implementation conventions, that continue to evolve and grow with every new application (Myers and Pohl 1994). Each implementation results in refinements, additions, deletions, and revisions of not only the physical parts but also the concepts and conventions embodied in the ICDM framework. In this respect each ICDM implementation is an experiment that yields a rich set of results for future implementations.

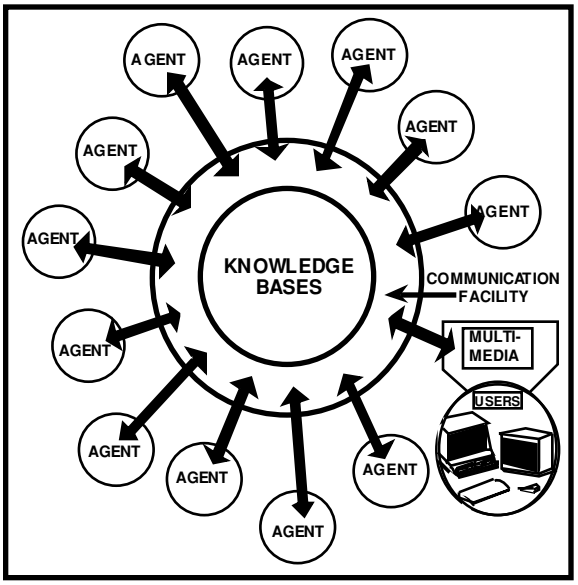


Figure 31: The ICDM Architecture

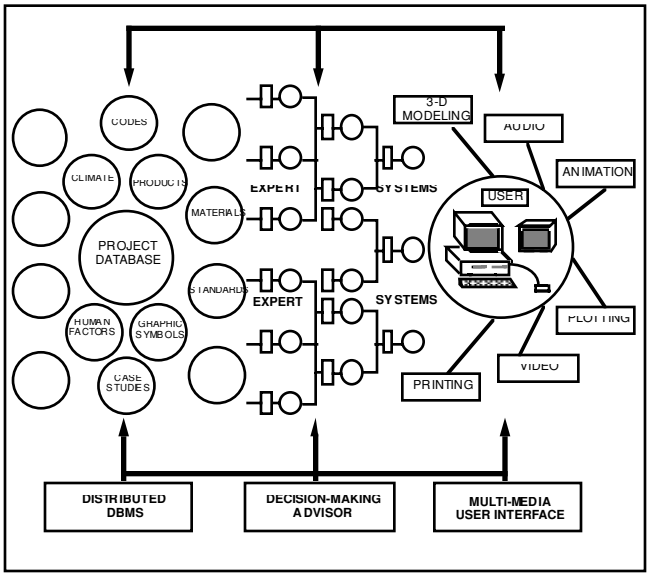


Figure 32: Principal ICDM Components

The decision-making advisory facility, shown schematically in the center of Figure 32, is the active core of any ICDM implementation. While the specific implementation design details may vary from application to application, from a conceptual viewpoint it

typically consists of several computer-based agents that interact with each other and the user(s) in a collaborative fashion. It is the primary responsibility of the agents to assist the human decision-maker(s) by monitoring, interpreting, testing, and evaluating the current solution state, and also by proposing solution strategies and alternatives. The current state of the solution and the problem context is normally held in memory, in some suitable format such as a semantic network of frames or objects that can be readily accessed by the agents. Although, in early implementations of the ICDM framework, the agents were mostly constructed as rule-based expert systems (Figure 33) this is not a requirement. Today, we typically take advantage of the flexibility and development speed provided by rule-based shells such as CLIPS (NASA 1992) for rapid prototyping and utilize object-oriented languages such as C++ (Stroustrup 1987) for end-user products.

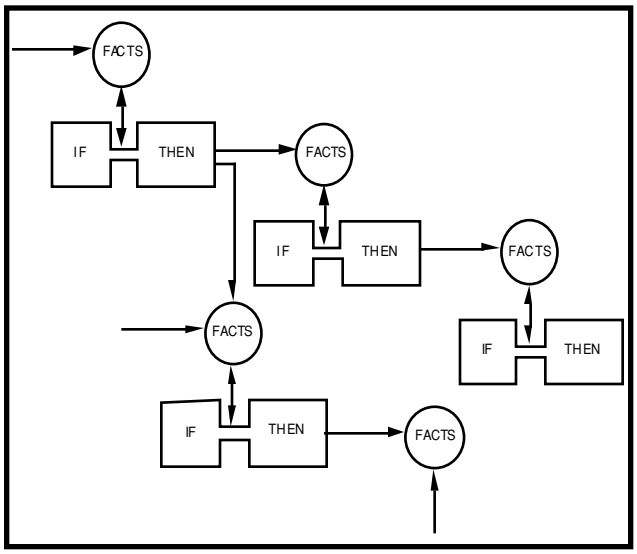


Figure 33: Rule-Based Agent Format

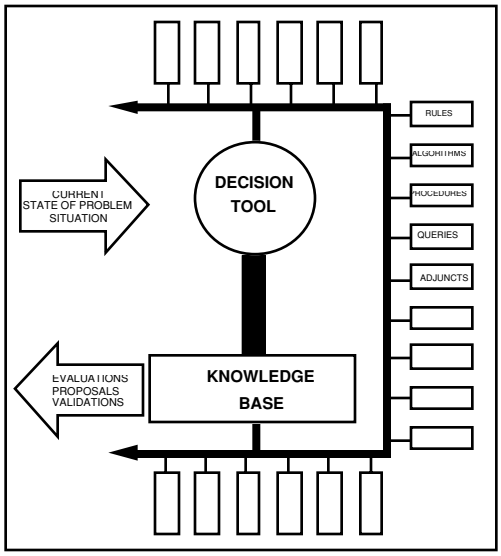


Figure 34: Conceptual Structure of an Agent

The agents are connected to the current state of the problem through an object-serving facility (i.e., referred to as Communication Facility in Figure 31) that automatically sends changes in the problem state to the appropriate agent, and allows agents to post the results of their actions onto the semantic network (Figure 35). Only those changes in information that are relevant to the responsibility domain of a particular agent are sent to that agent. This filtering is usually accomplished in a simple manner by requiring each agent to register with the system the type of information that it wishes to deal with.

In the case of service-agents, which are typically domain specific, the number and capabilities of the agents required in a particular domain depend largely on the level of detail involved in a given problem-solving task. For example, during the early stages of military mission planning when higher level conceptual warfighting strategies are being formulated, the combat support services element (CSSE) may be represented by a single agent capable of developing an overall logistical supply plan. During subsequent stages and in particular during the execution stage, the more detailed CSSE monitoring and support requirements will involve an increasing number of agents each responsible for a specific subset of the logistics problem area. In other words, the more detailed the

problem-solving task the more specialized the knowledge requirements of the collaborating agents. However, regardless of the type of agent and the level of domain specialization, each agent incorporates a communication adjunct that allows it to receive and send messages (in object format), and any number of rules, algorithms and other constructs that allow it to execute the tasks that it was designed to perform (Figure 34). Typically, an agent receives information about the current state of the problem, evaluates that information subject to its functional capabilities and knowledge, and sends the results of its deliberations back to the semantic network. Once these agent results have been posted on the semantic network they are immediately made available to other interested agents.

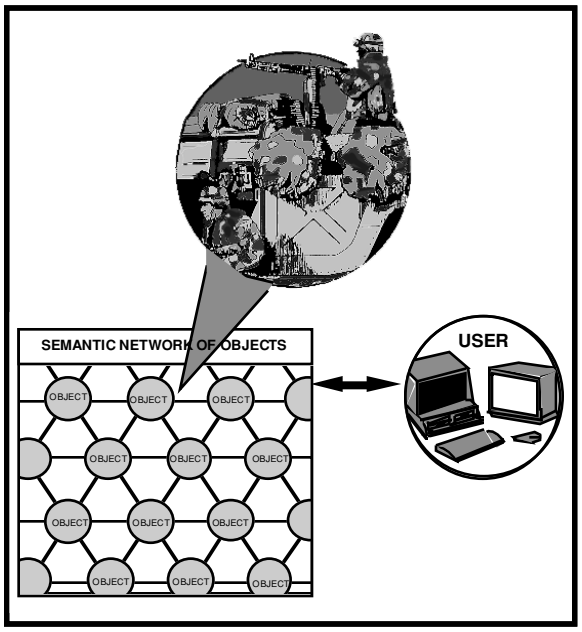


Figure 35: Semantic Network of Objects

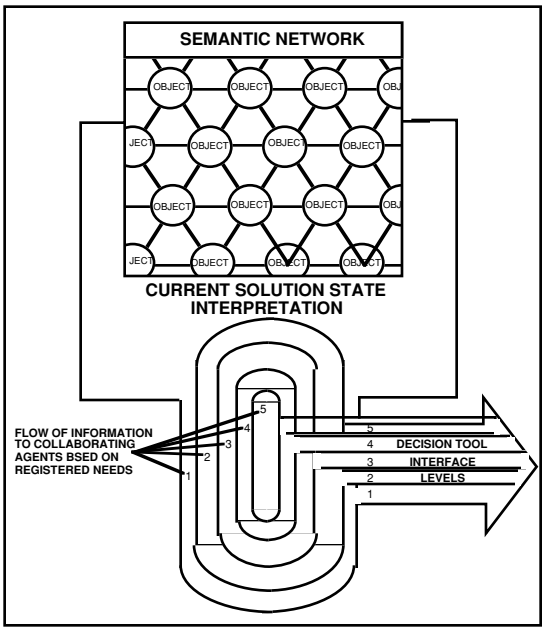


Figure 36: Information Flow to Agents

The ICDM model assumes the existence of some form of coordination facility within the decision-making advisory component. This facility may include one or more conflict identification and/or resolution agents operating in either a centralized or distributed mode. To what extent conflicts should be identified or resolved depends on the application, and is not in any way dictated by the ICDM framework itself.

The operation of a typical ICDM advisory facility can be described as follows. The current state of the problem situation (i.e., context and evolving solution) is represented in terms of high level objects, such as aircraft, armored vehicle, observation post, or enemy unit, and their relationships, in a semantic network (Figure 35). Each agent receives, on a continuous basis, information updates from the semantic network. However, not all available information is sent to all agents. A particular agent receives only the information changes that relate to the template of information requirements that the agent has registered with the system at that time (Figure 36). We look upon this as a subscription service which maintains a subscription profile for each subscribing client. Agents process the information updates as they become available (Figure 37). If an agent has already

started to work on information that is subsequently superceded by new information, then the agent will abandon the current task at the first opportunity and recommence the task with the new information. Our experience with past implementations of the ICDM framework has shown that in a real-time execution environment with preemption (i.e., interrupt) capabilities the possible posting of results based on obsolete information, by an agent, is acceptable. If the system is sufficiently responsive to changes in the current state of the problem represented in the semantic network, then it will correct itself within one or two message cycles.

There are two other advantages of the 'subscription service' concept that are related to the underlying communication transmission facility. Particularly in military command and control applications relying partly on wireless communication systems we must expect severe bandwidth limitations and a high degree of operational unreliability. For example, the EPLRS (Enhanced Positioning Location and Reporting System) and SINCGARS (Single Channel Ground and Airborne Radio System) currently available to the Marine Corps provide effective bandwidths of only 1000 and 300 b/sec, respectively. First, only the changes in information need to be transmitted since each client incorporates context information in its semantic network. Second, the object representation allows the transmissions to be optimized to some degree.

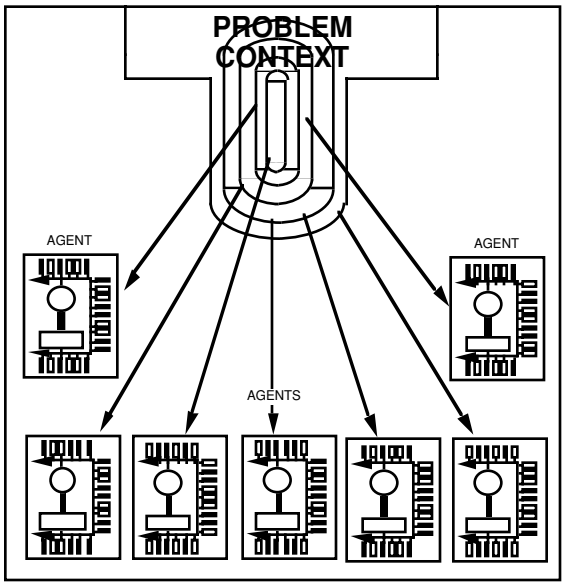


Figure 37: Agents Receive New Information

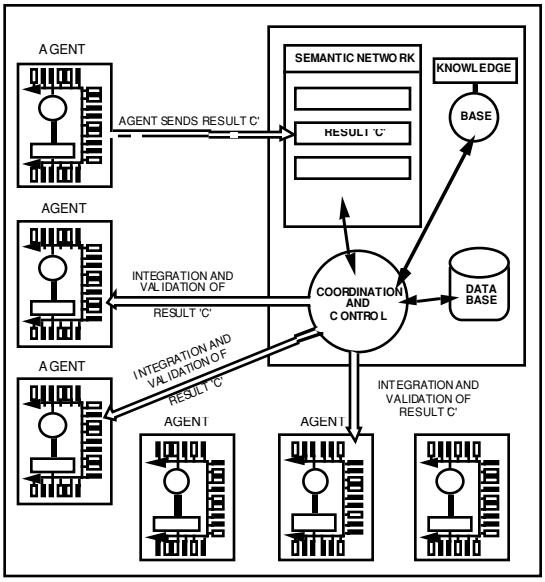


Figure 38: Agents Collaborate

The agents post their results on the semantic network where these results are now available to all other agents that have interest in the new information. Among these might be a 'conflict identification' agent that monitors disagreements among agents. In past implementations of the ICDM framework it has been found preferable not to assign veto power to a conflict identification and/or resolution agent, but allow the latter to enter into an iterative dialog with the participating agents (Figure 38). The final action precipitated by this dialog may be any one of the following:

- Consensus is reached among the agents and the basis of the agreement is implemented. In our experience we have found this to be the most common outcome. In the practical implementation of multi-agent systems there are several ways in which non-convergence can be detected and dealt with. Foremost among these is the availability of the user to adjudicate and impact the behavior of agents by setting parameters and priorities.

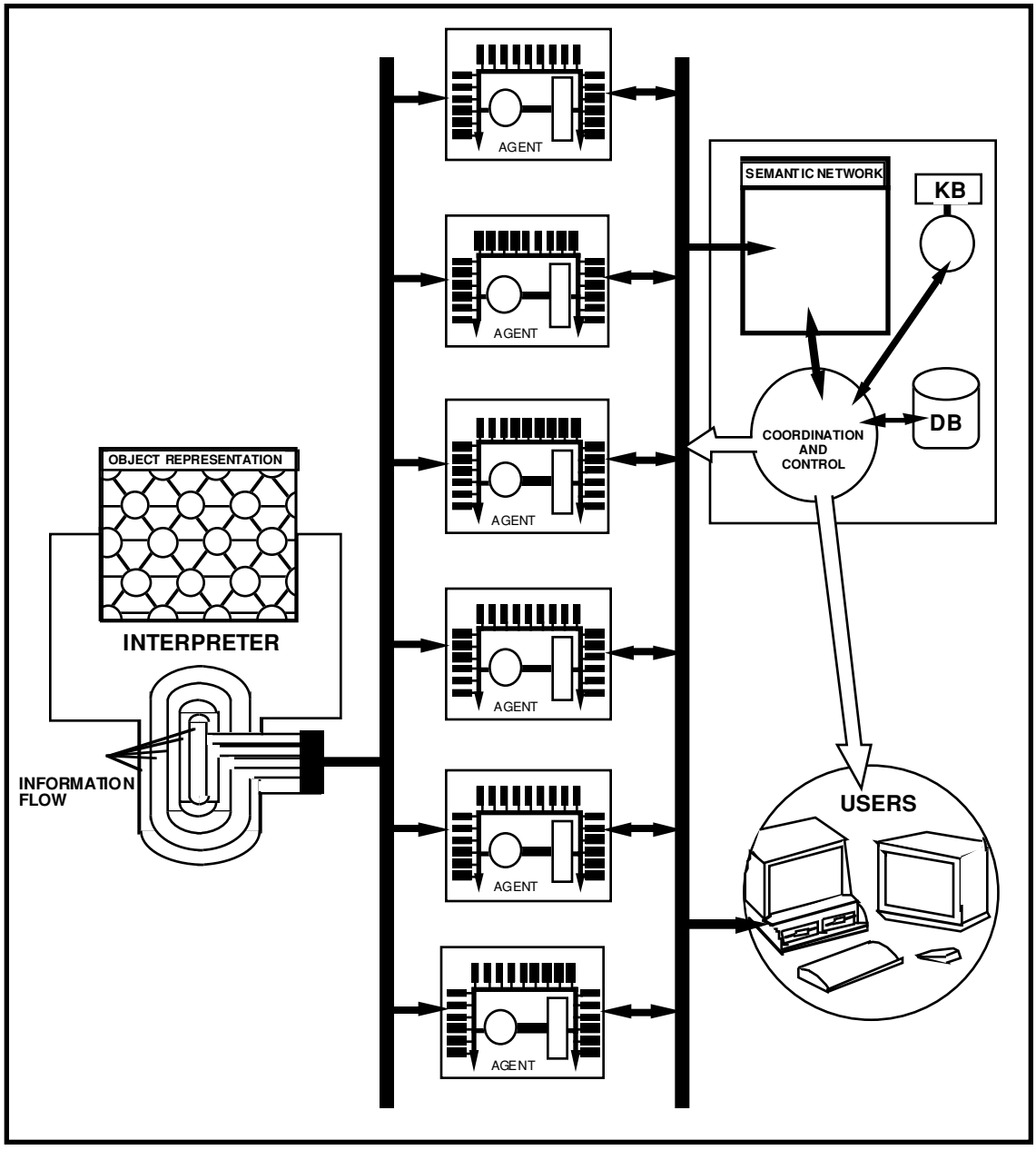


Figure 39: Conceptual View of a Typical ICDM Implementation

- The agents agree to disagree. In other words, acceptable tolerances of disagreement can be built into the collaboration facility. For example, three agents may agree on the existence of a single enemy target although each agent has a slightly different value for the location of the target.

- Disagreement on some issue by several agents is detected by a conflict identification agent. The latter joins the discussion and either facilitates an acceptable resolution of the conflict, or imposes a resolution strategy on the agents. The imposition of a resolution may be acceptable in some decision-support system applications under routine conflict conditions.
- The agents disagree and the conflict identification agent brings the situation to the attention of the user, outlines the facts of the disagreement, and proposes a possible courses of action to be taken by the user.

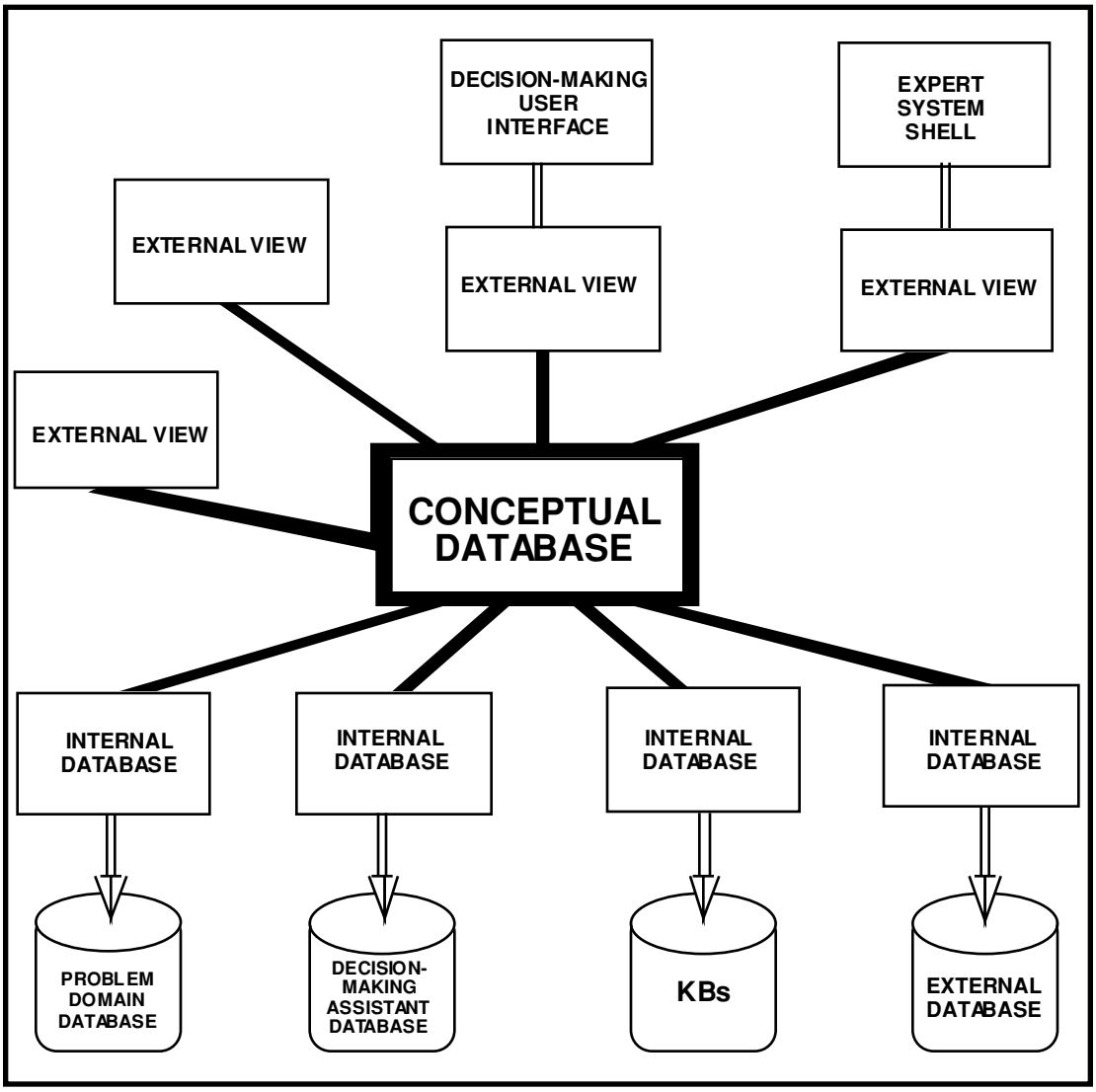


Figure 40: Integration of Multiple Heterogeneous Databases

A conceptual, integrated view of the functional elements of the ICDM collaboration and coordination facility described above is shown in Figure 39. In summary, the current state of the problem situation is communicated to the agents subject to the information interests currently registered by each agent with the coordination facility. The agents

process the information that they receive, in near real-time, within the context of their individual domains and objectives. The results of their deliberations are transmitted to the coordination facility where they are posted on the semantic network for consideration by the other agents. In the ensuing dialog among the agents conflicts are identified, discussed, and resolved with or without user interaction. The purpose of the ICDM model is to provide a development framework for cooperative and collaborative parallel decision-support applications. Although each application may call for a somewhat different implementation of the framework, the characteristics of concurrency, multi-tasking, near real-time responsiveness, user interaction, and high level representation, remain as common threads.

While many aspects of the ICDM model have evolved and changed over the past several years, the CADRC's earliest conviction that human agents should play a major role in any complex problem situation has grown progressively stronger. The problem solving activity presumes an element of the unknown, a problem that has to be solved through a decision-making process that cannot be completely predefined because of incomplete information and dynamic information changes. Under such conditions, the ability of the human partner to apply intuition (see Section 3.2.6) is arguably a necessary complement to the logical capabilities of the computer-based agents.

4.2 Desirable Database Management Capabilities

One of the early targets of multi-agent systems is the integration of existing information sources (i.e., databases and stand-alone, legacy programs) into comprehensive decision-support systems. The initial focus of such efforts is the linkage of existing databases. This is not a trivial task since these existing information resources typically were implemented in many different ways. Consequently, any integrating system (including a multi-agent system of the kind described here) will be required to support the conceptual integration of a variety of data resource models. This can be accommodated through the provision of several internal-level database representations, requiring a number of additional mapping functions to link the internal and conceptual representation levels (Figure 40). In this way, any externally linked database can be removed or replaced by another database, simply by recoding the internal to conceptual level mapping. Since this will not affect the external data representation, the user-interfaces built at the external level will also remain unchanged.

The scope of database query facilities desirable for the kind of multi-agent, decision-support environment discussed here far exceeds traditional database management system (DBMS) functions (Figure 41). They presuppose a level of embedded intelligence that has not been available in the past. Some of these desirable features include: conceptual searches instead of factual searches; automatically generated search strategies instead of predetermined search commands; multiple database access instead of single database access; analyzed search results instead of direct (i.e., raw) search results; and, automatic query generation instead of requested searches only.

A traditional DBMS typically supports only factual searches. In other words, users and applications must be able to define precisely and without ambiguity what data they require. In complex problem situations users rarely know exactly what information they require. Often they can define in only conceptual terms the kind of information that they are seeking. Also, they would like to be able to rely on the DBMS to automatically broaden the search with a view to 'discovering' information.

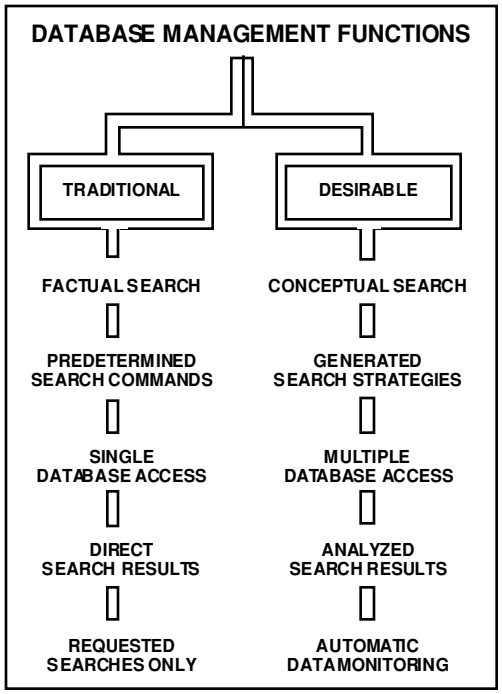


Figure 41: Comparison of DBMS Features

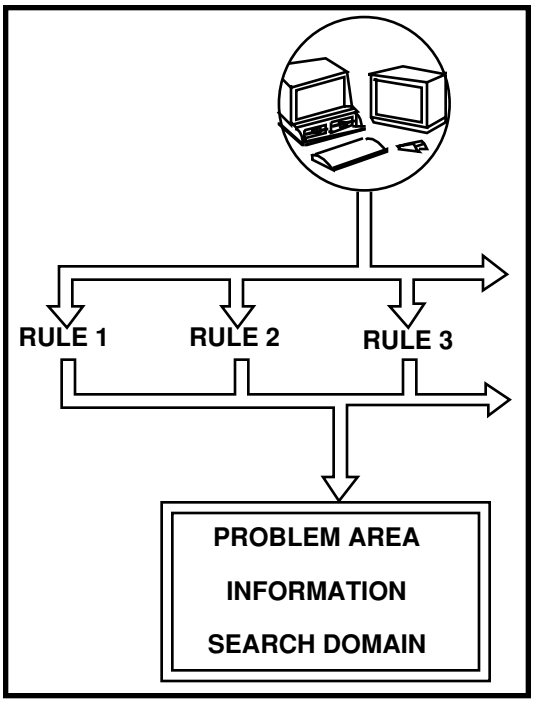


Figure 42: Conceptual Query Management

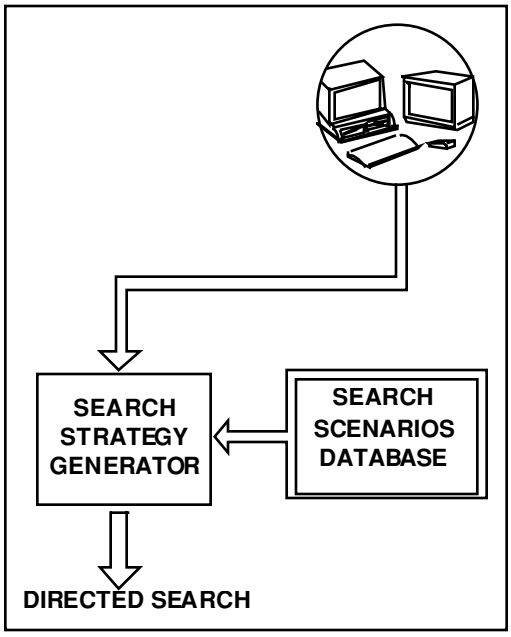


Figure 43: Search Strategy Generation

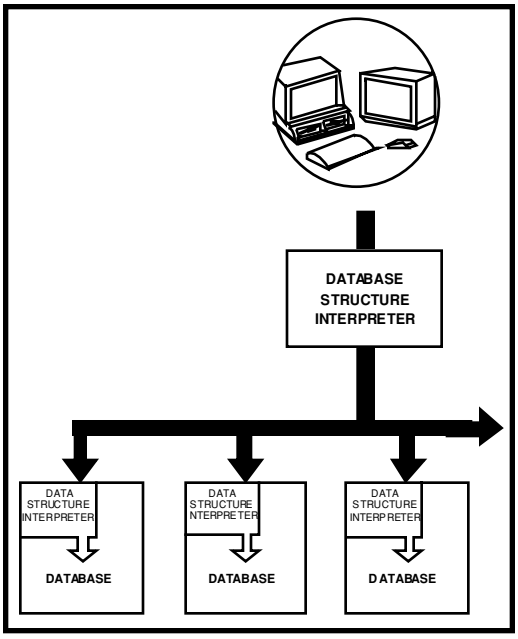


Figure 44: Multi-Database Access Management

This suggests, in the first instance, that an intelligent DBMS should be able to formulate search strategies based on incomplete definitions. It should be able to infer, from rather vague information requests and its own knowledge of the requester and the problem context, a set of executable query procedures (Figure 42). To facilitate this process the DBMS should maintain a history of past information requests, the directed search protocols that it generated in response to these requests, and at least some measure of the relative success of the entire search operation (Figure 43).

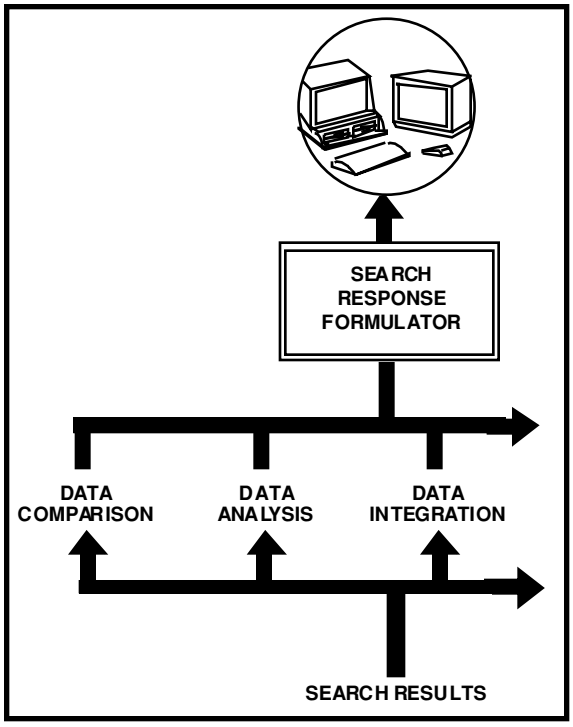


Figure 45: Multiple Source Search Results

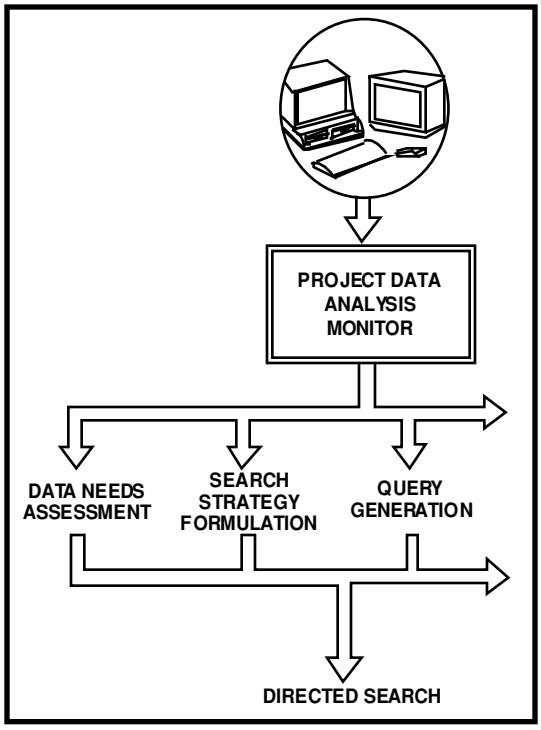


Figure 46: Automatic Query Management

A traditional DBMS normally provides access to only a single database. A knowledge-based decision-support environment is likely to involve many information sources, housed in a heterogeneous mixture of distributed databases. Therefore, through the internal-level database representations discussed earlier (Figure 40), the DBMS must be able to access multiple databases. Using the mapping functions that link these internal representations an intelligent DBMS should be capable of formulating the mechanisms required to retrieve the desired data from each source, even though the internal data structures of the sources may differ widely (Figure 44). Particularly when search results are derived from multiple sources and the query requests themselves are vague and conceptual in nature, there is a need for the retrieved information to be reviewed and evaluated before it is presented to the requester (Figure 45). This type of search response formulation facility has not been necessary in a traditional DBMS, where users are required to adhere to predetermined query protocols that are restricted to a single database.

Finally, all of these capabilities (i.e., conceptual searches, dynamic query generation, multiple database access, and search response formulation) must be able to be initiated not only by the user but also by any of the computer-based agents that are currently participating in the decision-making environment. These agents may be involved in any number of tasks that require the import of additional information from external databases into their individual knowledge bases (Figure 46).

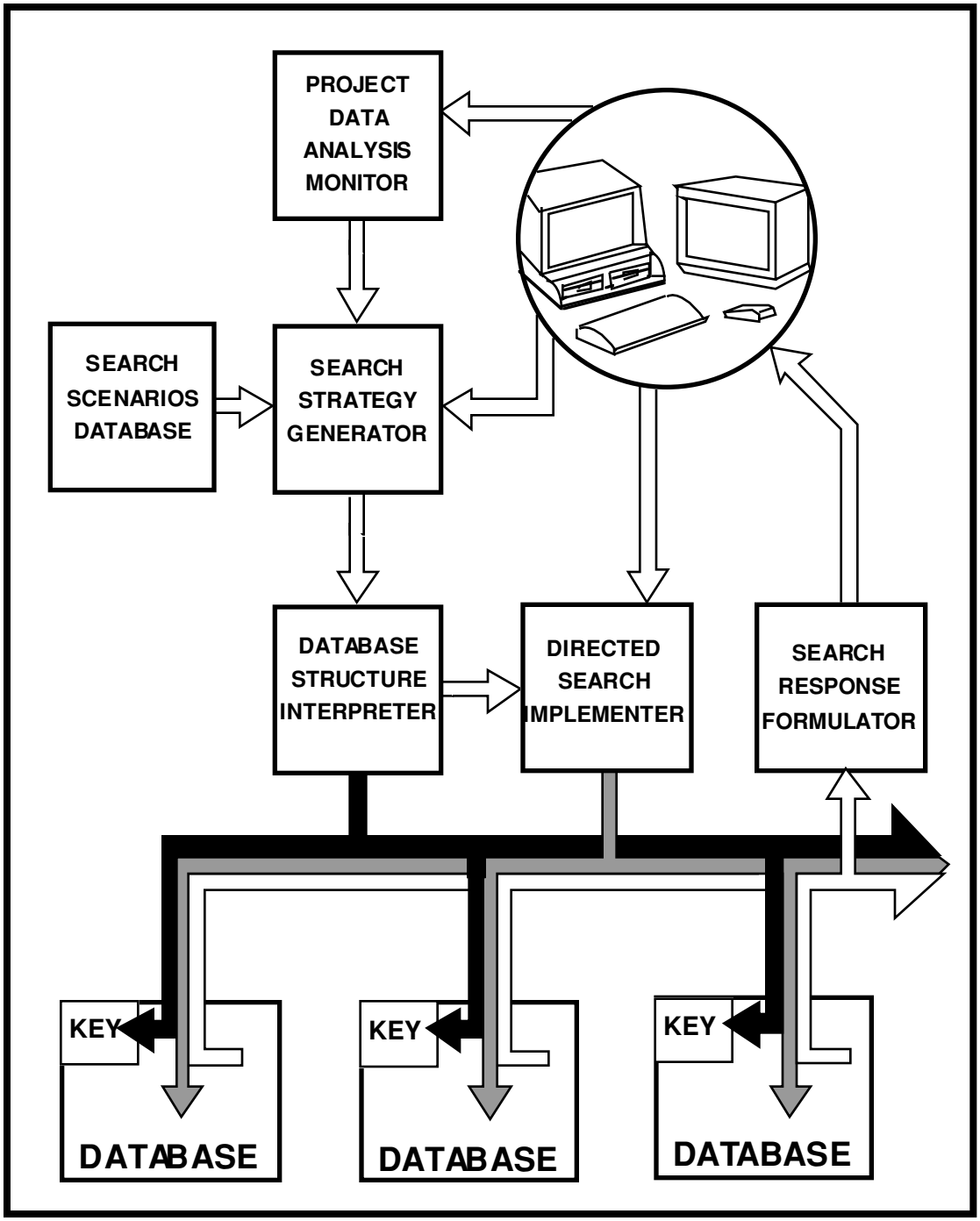


Figure 47: Conceptual Model of an Intelligent DBMS Interface

A conceptual model of an intelligent DBMS interface with the capabilities described above is shown in Figure 47, and forms the basis of the following typical information search scenario that might occur in an integrated and distributed, collaborative, multi-agent, decision-support environment. Queries that are formulated either by the user or generated automatically by a computer-based agent are channeled to a Search Strategy Generator. The latter will query a Search Scenario Database to determine whether an appropriate search strategy already exists from a previous search. If not, a new search strategy is generated, and also stored in the Search Scenarios Database for future use. The search strategy is sent to the Database Structure Interpreter, which automatically formulates access protocols to all databases that will be involved in the proposed search. The required access and protocol information, together with the search strategy, are sent to the Directed Search Implementer, which conducts the required database searches. The results of the search are sent to a Research Response Formulator, where the raw search results are analyzed, evaluated and combined into an intelligent response to be returned to the originator of the query.

The proposition that the DBMS interface should be able to deal with incomplete search requests warrants further discussion. When searching for information, partial matching is often better than no response. In traditional query systems, a database record either matches a query or it does not. A 'flexible query system', such as the human brain, can handle inexact queries and provide best guesses and a degree of confidence for how well the available information matches the query (Pohl et al. 1992 and 1994). For example, let us assume that a military commander is searching for a means of trapping a given enemy force in a particular sector of the battlefield and formulates a '*something like* a choke point' query. In a flexible query system a '*something like*' operator would provide the opportunity to match in a partial sense, such as: terrain conditions that slow down the movement of troops; unexpected physical obstacles that require the enemy to abruptly change direction; subterfuge that causes enemy confusion; and so on. These conditions can all, to varying extent, represent '*something like*' a choke point that would be validated by a degree of match qualification.

Flexible query processing systems are fairly common. For example, most automated library systems have some level of subject searching by partial keyword or words allowing users to browse through a variety of related topics. Even word-processing programs include spelling checkers, which by their very nature search for similar or related spellings. However, even a flexible query system cannot automatically form hypotheses, since the system does not know what to ask for.

The ability to search for '*something like*' is only a starting point. How can the system be prompted to search for vaguely or conceptually related information? For example, how can the system discover the intuitive connection between a physical choke point, such as a narrow cross-corridor in a mountainous battlefield, and a precision fire maneuver aimed at concentrating enemy forces in an exposed area. In other words, how can the system show the commander that the precision fire maneuver option can satisfy the same intent as the cross-corridor option. In addition, the system must not overwhelm the commander with an unmanageable number of such intuitive speculations. To discover knowledge it is

necessary to: form a hypothesis; generate some queries; view and analyze the results; perhaps modify the hypothesis and generate new queries; and, repeat this cycle until a pattern emerges. This pattern may then provide insight and advice for intuitive searches. The goal is to automate this process with a 'discovery' facility that repeatedly queries the prototype knowledge bases and monitors the reactions and information utilization of the problem solver, until knowledge is discovered.

5. Military Decision-Support Applications

The information society is bringing changes that are being felt in all areas of human endeavor. It is particularly evident that these changes are coming in rapid succession and that many of the methods and systems that commerce, industry and government have relied on in the past to deal with change and to support complex tasks are becoming less and less effective.

Typically, major changes are made only when they can potentially bring major gains, or if failure to change will result in severe penalties. In either case, major changes take time. They usually require cultural changes and are best implemented incrementally. The implementation process is complicated by the human behavioral tendency under conditions of stress, such as a battlefield, to revert to the old paradigm. Clearly, therefore, changes are unsettling. They require the human cognitive system to adapt to new conditions, to reevaluate relationships, beliefs and solution strategies. In short, as discussed previously (see Section 2.2), this forces the human being into a situation where past experience tends to lose some of its value and risks have to be taken to develop new understandings and generalizations to form the basis for new experience.

Viewed from an optimistic stance, changes provide new opportunities. In fact, the more changes the more opportunities. These opportunities are not restricted to technical advances and solutions, but are equally prominent in management areas that involve the organization and orchestration of groups of persons to collectively accomplish a goal. Integration, cooperation, persuasion, and motivation are key elements that acquire new meaning in a rapidly changing environment. The military services, in particular, are confronted with societal and technical changes that will have profound influence on the design, implementation and operation of their planning, execution and training systems. In reference to Figures 48 and 49, prominent among these changes are the following:

- Increasingly, the US military forces will be based in the Continental United States (CONUS), although most of their missions will take place outside CONUS. This places a great deal of emphasis on rapid deployment capabilities, as well as integrated, reliable communication and coordination facilities.

- Increasingly, the US military forces will be involved in joint operations, often involving foreign countries as allies. This requires an unprecedented level of coordination, flexibility, interconnectivity, global sensitivity and knowledge (e.g., foreign languages), training, and standardization.

- Increasingly, the value of human life is becoming a major factor in US military operations. Casualties are a liability with severe political repercussions. Effective protection of the warfighter is a difficult and expensive proposition.

- A large percentage of missions, whether humanitarian or military, will involve smaller forces that may be widely dispersed. Small units (e.g., squads of six to eight soldiers) and individual soldiers will be required to perform functions that are currently accomplished by specialized support personnel. Effective, fail-proof communication, near real-time collaboration and coordination, and computer-based, intelligent decision-making assistance are essential prerequisites for such missions.

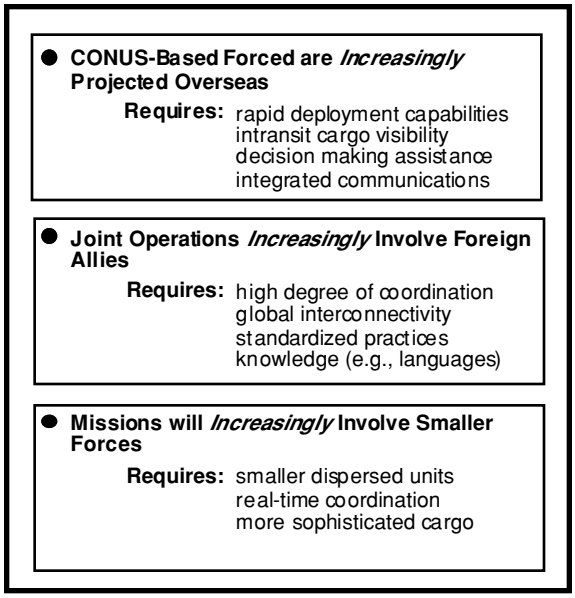


Figure 48: Changing Military Context

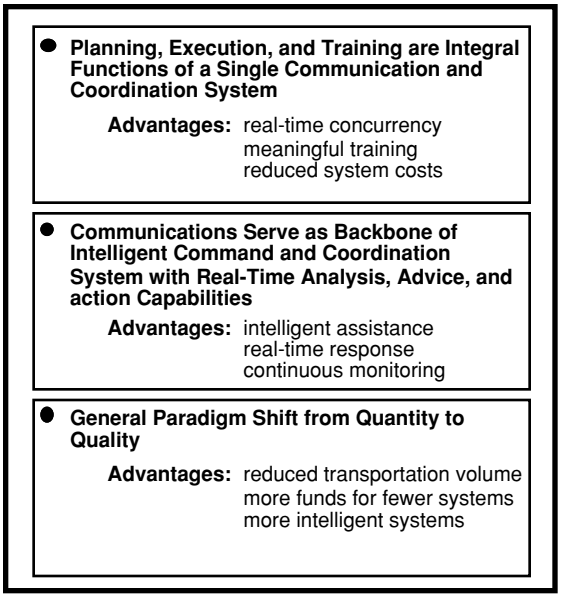


Figure 49: Emerging Infrastructure Notions

- Planning, execution and training are being increasingly viewed as integral functions that must be supported in one holistic communication and collaboration system. The traditional separation of these functions into distinct systems is expensive, fragmented, time consuming, and inefficient.

- There is an increasing recognition that communication should not be restricted to chain of command protocols (US Army 1994). Data must be available where needed, rather than controlled by ownership. This suggests a 'network' rather than 'hierarchical' approach to the design and implementation of communication systems.

- Increasingly, computer-based decision-support systems are being viewed as intelligent assistants in an interactive, near real-time human-computer partnership. This is a direct outcome of the greater complexity of global scale problems, the greater emphasis on rapid deployment, the need for the small unit or individual soldier to respond rapidly to changing conditions when operating in a dispersed mode in the theater, and the greater reliance on individual initiative at all levels and in all direct and indirectly supportive mission tasks.

- There is a general shift from 'quantity' to 'quality' mandated by decreasing budgets, the need to deploy rapidly, the increasing value placed on human life, and the political impact of continuous media coverage.

Military missions and all of their supportive operations are by nature complex problems involving intricate relationships among many variables, under conditions of uncertainty. During all phases they are subject to dynamic information changes that impact both the solution objectives and the strategies for orchestrating solutions. Decision-makers at every level must be able to understand and respond quickly to changing circumstances, and this applies equally to planning and execution. Training becomes an integral component of the decision-making process, allowing the decision-maker to simulate, explore and experiment prior and during the actual operation.

5.1 Driving Forces, Responses, and Opportunities

The US Department of Defense (DoD) is moving aggressively to replace old methods and decision-support systems with new concepts and systems that reflect current and projected future societal changes and technological advances. In particular, DoD is intent on leveraging the opportunities that these changes and advances offer. Such efforts cannot be confined to prudent planning activities that every organization undertakes on a routine basis, but must respond to past and present experiences that demonstrate increasing difficulties with the status quo. For example, decreasing budgets require 'downsizing'. While 'downsizing' may have mostly negative connotations when viewed within the context of past beliefs, it offers important opportunities for improving the effectiveness of an organization and the quality of its services and products.

In recent years DoD has instituted several major programs aimed at taking advantage of such opportunities. For example, during the past two decades DoD has experienced a proliferation of computer-based decision-support systems in the tactical, logistical, and administrative areas. Each of these systems requires funding for maintenance and support, and most also receive substantial funding for development and enhancement purposes. In most cases these systems were proposed and developed in response to specific needs, by functional groups. Accordingly, emphasis was placed on meeting the specific needs of the functional group rather than the overall system requirements of the organization. Despite the efforts of agencies charged with coordinating these individual system developments to ensure that they would conform to system-wide plans, standards, and protocols, an acceptable level of integration has not been achieved. In fact, the coordinating agencies have found it increasingly difficult to orchestrate the proliferating systems toward an integrated global system.

To overcome this mounting problem DoD initiated a major program to evaluate existing computer-based systems and select a smaller number of 'migration' systems that would accommodate all of the required functions and replace the other systems. This is an important initiative with significant potential operational and economic benefits. In particular, this action can lead to a major reduction in funding requirements while

increasing the funding resources that are available for the remaining 'migration' systems. A simple calculation can demonstrate this point.

Let us assume that there are currently about 40,000 (a very conservative estimate) major computer-based decision-support systems used by the US military services. If we further assume that the average cost of maintenance and development is $1 million per year for each system, then the total annual direct cost of supporting these 40,000 major software applications is $40 billion. Now, if after evaluation, these systems are reduced to 8,000 'migration' systems and even if we triple the annual maintenance and development funding for each of the 'migration' systems to $3 million, then a net saving of $16 billion (40%) has been achieved. However, potentially, a great deal more will have been accomplished. The relatively small number of remaining systems can be extended, redesigned or replaced with the additional funding that has become available, to achieve a much higher level of integration and functionality. Furthermore, the resulting integration effort provides an opportunity for incorporating recent technological advances and system design concepts into the system architecture. In particular, it becomes possible to redesign the overall system architecture to be more amenable to technological advances and societal changes in the future.

From a more general point of view, there are several concurrent forces that are converging in similar directions. The US Army and the US Marine Corps are currently reexamining the way in which they propose to conduct military operations in the future. Examples of these efforts are the Army's Force XXI (US Army 1995), the Marine Corps' Sea Dragon (Krulak 1996 and Bergman 1996), and the Small Unit Operations (SUO) task force of the Defense Advanced Research Projects Agency (DARPA). In all cases these initiatives are aimed at leveraging technological advances within the realities of a changing world context to achieve more with less.

- Smaller, but more effective, forces in the theater; widely dispersed to maintain control over larger areas; superbly equipped and well supported through intelligent communication and coordination systems; capable of exercising initiative and generating intelligence data.

- Integrated, intelligent communication and collaboration systems that are not only capable of message passing and data collection, but are also capable of analysis, reasoning, generating alternatives, determining consequences, advising, and learning.

- Faster logistical deployment with continuous visibility of in-transit equipment and supplies.

- Removal of the physical presence of human warfighters and support personnel from all situations where they can be replaced by remote communication and unmanned war machines.

- Better protection of the warfighter through protective uniforms, reactive armor, noise and vibration mitigation, deception and cloaking, increased fire

power, vision and surveillance support, real-time access to advice and expertise from remote sources, flexible and fail proof communication, fail proof navigation devices and support.

- Greater emphasis on training that is meaningful, based on real world situations, and can be accessed by the individual soldier at any time in any location (including the theater).

5.2 Desirable System Features

Several system requirements are readily discernible from the foregoing discussion. Foremost among these is the need for integration. Neither the tactical nor logistical support needs of Force XXI or Sea Dragon can be met without full integration of command, communication, coordination, documentation, intelligence, and training functions within one system umbrella. This umbrella system must have an open architecture that allows growth and provides flexibility. It must facilitate the addition of new modules and the replacement of existing modules in a manner that is transparent to the users. These requirements point to a distributed architecture, incorporating object-oriented concepts, with a great deal of internal connectivity and some degree of redundancy.

The system must support a high level of parallel activity, even within the modules supporting a particular functional area. The need for near real-time response capabilities is incompatible with large deep simulation software packages that have to run their full course once they have received the necessary input, and cannot be halted and/or redirected as soon as new information that would change this course becomes available. A cooperative system architecture that allows many smaller modules to continuously interact with each other and collectively contribute to the decision-making process is more likely to satisfy these response needs.

The need for users to interact with the system at all levels and under many different circumstances, ranging from headquarters tactical planning to logistical execution to communication with the warfighter in the theater, necessitates artificial intelligence. As discussed previously (see Sections 1.4.5 and 3.2.2) a prerequisite for embedding artificial intelligence capabilities in a system is the presence in that system of a high level representation of the real world objects that the user reasons about. Graphical images of theater maps that are not represented in the computer system as objects (i.e., roads, buildings, rivers, enemy units, etc.) are empty shells that cannot be used by the system to automatically analyze consequences, generate alternatives, and provide advice. The lack of a high level representation is the most serious limitation of the vast majority of software systems used by the US military services today.

Military users are increasingly suggesting system requirements, such as intuitive interfaces, near real-time advice, analysis and interpretation capabilities, and virtual reality training facilities, that indicate a desire for a human-computer partnership rather than automation. Within the context of current technological capabilities and anticipated near

term advances, this desire can be satisfied. This does not mean that we can build systems today that emulate human behavior to a level that is comparable with actual human capabilities. What we can do is to design and implement a system architecture that will support an evolving artificial intelligence capability. Even in its infant form this capability will represent a substantial advancement over existing system capabilities, and over time it will become more and more powerful and sophisticated.

The system must integrate functional divisions that are based on historical roots rather than actual performance objectives and requirements. In the past planning, execution, and training functions have been treated as largely separate endeavors and have been accommodated in separate systems. These divisions are incompatible with the increasing need for near real-time response to changing conditions, flexibility, faster deployment, and continuous access to realistic training environments. The accelerating rate of change precludes the luxury of planning and implementing in discrete, sequential steps. Instead, planning and execution will merge, more and more, into parallel activities. Under these circumstances the continued separation of planning and execution functions into separate systems will become a serious source of delay, disruption, and ultimately failure.

The case for integrating training with planning and execution is just as strong. The purpose of training is to prepare persons to deal effectively with future situations. If the future situation can be accurately predicted and there is sufficient time to formally describe and present this situation to the trainee, then there is no compelling reason to integrate training with planning and execution. However, in a changing environment prepackaged training tends to lag well behind the actualities of application. The greater the rate of change in the application arena, the further removed from reality and the less effective the training becomes. When the trainees recognize this disparity between what is being taught and what can be expected to occur, they become less and less interested in pursuing any of the prepackaged training activities. As a result the training program loses its effectiveness and the trainees are poorly prepared for the tasks that they are expected to execute under real world conditions.

Apart from the performance advantages, the integration of planning, execution, and training also offers potential cost benefits. Although a system incorporating three functional areas is intrinsically more complex than a system that accommodates only one functional area, it greatly reduces the tendency for duplication both in respect to software functionality and hardware facilities. For example, the need to generate real world context and data in a separate training system is a significant and costly requirement. If the training function is integrated with the planning and execution functions then the real world context and data also serve as the training environment.

5.3 Databases and Applications as Shared Resources

The existence of a global communication network provides opportunities for integration that have not been available in the past. It allows integration to be addressed from a much more general point of view, in respect to information sharing and accessibility rather than the linkage of individual software applications (i.e., computer programs). In this view a

database is not owned by any particular computer program, but is treated as a resource that is available to any number of programs that are authorized to access this information resource. The actual geographical and physical location of either the database or the program that wishes to access the information contained in the database is immaterial.

However, there are two requirements that are of critical importance in this view of a database as a shared resource. First, the information stored in each database must conform to a common vocabulary. This ensures that the users of the information (both human users and other computer programs) clearly understand the meaning of each individual data element, and provides a basis for avoiding ambiguity and duplication. Second, each information resource must make its data available in the format of an 'external view' (see Section 4.2 and Figure 40) that may differ significantly from the internal data storage structure of the resource. This obviates the need for imposing rigid standards on the internal structural format of each database. Past attempts to enforce such data storage standards have been singularly unsuccessful. However, data access and transmission standards (e.g., SQL, dxf, IGES, etc.) that provide an 'external view' of the data have been readily adopted by industry and commerce.

The information that is shared in a distributed communication environment is not limited to the data stored in databases. Individual software applications become generators of information that may be stored in databases or, more often, will be shared directly with other applications. In this respect a global command and control system can be viewed simply as consisting of a large number of sharable resources. Some of these resources are databases that serve as depositories of dynamically changing data. Others are software applications that analyze, evaluate and generate views of combinations of data that are of interest to users. The following advantages of such a distributed, but integrated, cooperative command and control system are readily apparent:

- The notion of communication can be extended from the limited function of data transmission, to the much more powerful functions of processing data to information and applying artificial intelligence to assist in the cognitive processes that transform information into knowledge. The human decision maker working in partnership with the computer-based assistance provided by the collaborative system is now able to focus on the judgments that must be made to formulate decisions (Figure 50).

- The various software applications and their individual program modules can operate in parallel (see Section 3.1.2 and Figure 18), sharing databases and contributing the results of their analyses, evaluations and inferences to each other and the users. To take full advantage of this operational concurrency the program modules must be designed to respond to changes in data and new information in near real-time, opportunistically. For example, this means that the inter-process communication and collaboration facilities must provide preemption (i.e., interrupt) mechanisms, so that the individual program modules will automatically and

continuously realign themselves to the current state of the dynamically changing state of the problem situation.

- Hierarchically and sequentially controlled decision-making processes (see Section 3.1.1 and Figure 15) that tend to result in fragmentation and delays in 1st Wave software architectures are replaced by authorization protocols that maintain security without impeding the flow of information, which is critical to the decision-making environment. In a networked system of shared resources communication will not be constrained by chain of command. Necessary control is exercised through communication and analysis of the problem situation, rather than restricting the activities and contributions of personnel.

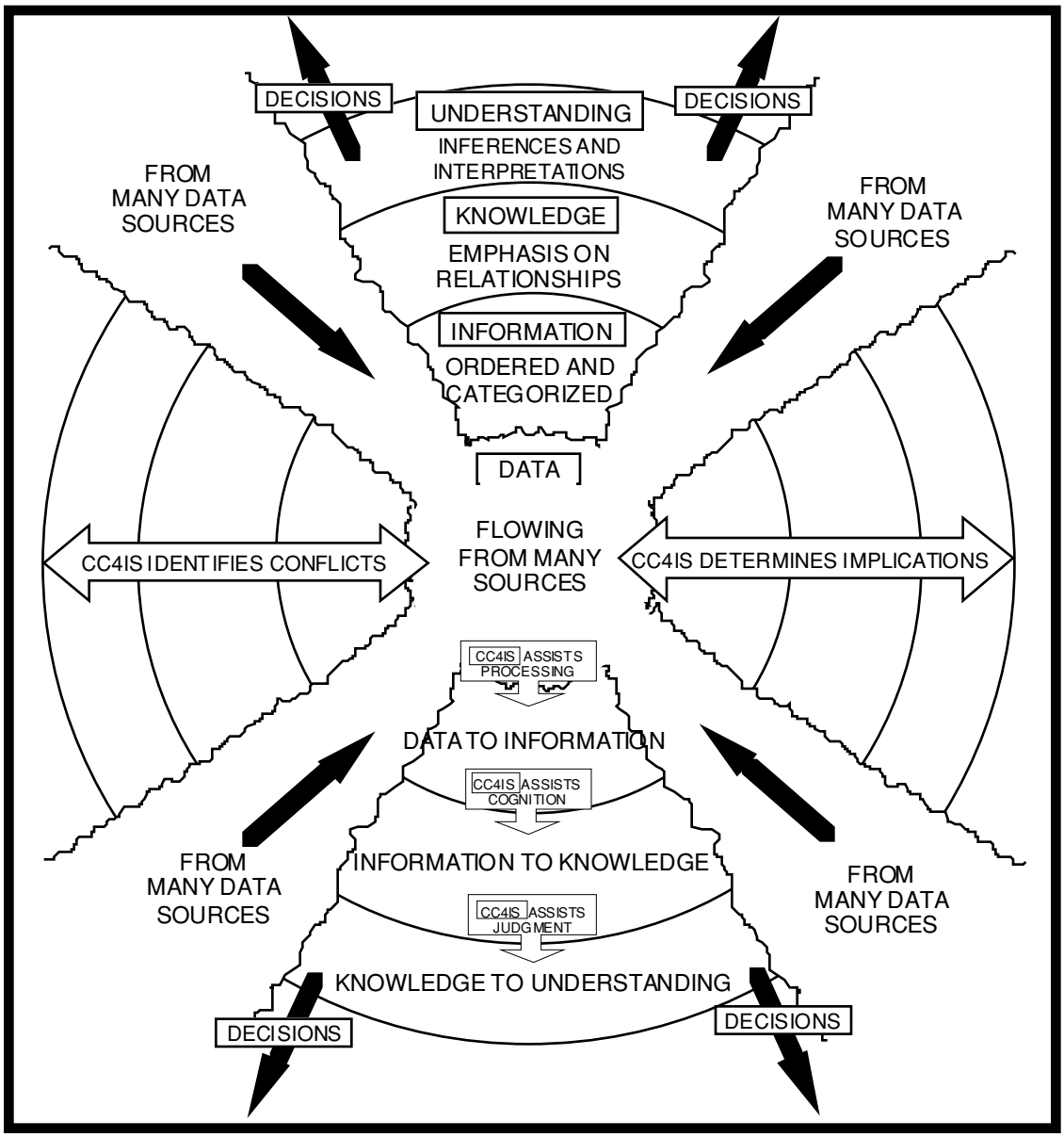


Figure 50: The Functions of a Cooperative Decision-Support System Within the Cognitive Model of Command and Control

- A great deal of meaningful and useful activity can take place at any node of a distributed communication and collaboration system, thereby encouraging the decentralizing of planning, execution and training functions. Decentralization is highly desirable for several reasons: reduction of communication and decision-making bottle-necks; accelerated planning and failure recovery through redundancy; and, capture of near real-time intelligence through strategically located local nodes.

In a distributed network of sharable resources planning, execution and training activities are supported by multiple program modules that are able to access the required databases and communicate with each other to exchange information and instructions that will initiate the execution of the desired functions. The program modules, therefore, must incorporate the necessary inter-process communication facilities that allow them to send and receive messages (e.g., information queries, results of evaluations, proposals, and requests for services, in an object-based format), transmitted through the communication network.

In this respect, the integration of the planning, execution and training activities is largely a user-interface issue. The user selects the desired mode of operation and specific data sources (if any), to initiate a sequence of internal system activities that seamlessly access those system resources that are required for the completion of the task. It is a relatively trivial matter to embed sufficient intelligence in the interface components for the system to automatically infer, based on limited user-interaction, whether the user desires to undertake a planning, execution or training task, and transmit the appropriate messages to the component that is capable of completing this task. These components may themselves consist of multiple program modules that are capable of communicating among themselves and with the outside world. In this regard, the collaborative command and control system architecture resembles a conglomerate of multi-agent systems (see Section 2.3.2 and Figure 9), and an extension of the 2nd Wave multi-agent software paradigm shown earlier in Figure 7 (see Section 2.3.2).

In each functional component the planning, execution and training functions are accommodated either by separate program modules or alternative modes of execution of the same module. For example, the differences between planning and execution modes may be accommodated both through internal software switches and the invocation of additional agents. When required to operate in a training mode, the appropriate component would first interact with the user to determine the objectives and level of training desired, select the appropriate training context, and then activate one or more agents to monitor and assist the user during the training session. Additional provisions can be made for evaluating the performance of the trainee and capturing portions of the training session for play-back and 'lessons learned' analysis.

6. ICODES: Ship Loading with Service-Agents

In 1994, the Collaborative Agent Design Center (CADRC) entered into a contract with the US Navy on behalf of the Military Traffic Management Command (MTMC) of the US Army to develop a ship stow-planning decision-support system that could meet the logistical planning requirements of large scale military deployments to overseas theaters. Experience during the Desert Storm offensive had shown that existing legacy systems, based on 1st Wave software technology (see Section 2.3.1), tended to perform poorly under surge conditions. In particular it was found that these systems did not allow the human decision-maker to react quickly to changing conditions, nor did they provide adequate assistance to less experienced users.

Ship stow-planning is the process of planning the loading, placement and unloading of cargo on ocean-going vessels and barges. Cargo stowage and mobilization planning expertise takes years to develop and the US Department of Defense (DoD) is losing this experience due to downsizing and retirement of personnel. MTMC, as DoD's single traffic manager for military cargo moving through the Defense Transportation Network (DTN), commissioned the CADRC to design and develop a knowledge-based planning system to provide mobilization and cargo planners with intelligent assistance throughout the stow-planning process.

6.1 Ship Stow-Planning as a Complex Problem

The rapid deployment of military assets from the US to overseas (OCONUS) locations is a complex undertaking. It involves the movement of large numbers of tracked and wheeled vehicles, weapon systems, ammunition, power generation and communication facilities, food supplies, and other equipment and goods, from military bases to the area(s) of operation. Several modes of transportation are typically involved. Depending on the location of the military base the assets are preferably moved by road to the nearest railhead, from where they are loaded onto rail cars for transportation to the port of embarkation (POE). Alternatively, if rail transportation is not an option, all of the cargo must be shepherded through the public road corridor from the base to the port. At the POE the assets are briefly assembled in staging areas and then loaded onto vessels for shipment. Points of debarkation (POD) may vary widely from a commercial shipping port with fairly good facilities to an amphibious landing on a hostile shoreline under fire.

Speed and in-transit visibility are of the essence (Figure 51). The total time required for the deployment becomes a critical factor in the development of the overall mission plan. There is a strong desire to reduce the in-transit time first from weeks to days, and then from days to hours. While faster ships can reduce voyage times by several days, the rapid movement of cargo through each node of the transportation infrastructure is equally important. This can be achieved only through sound planning, rapid response to changes, and a high degree of integration. At the same time there is a need for commanders to know precisely where their assets are located at any time, so that they are not unduly inhibited by logistical constraints in the modification of mission plans before and during execution.

This necessitates an integrated, communication-based documentation system that tracks each cargo item from origin to destination. Currently, MTMC supports this function through the Worldwide Port System (WPS), which transmits updated cargo lists and manifests from node to node within the transportation system.

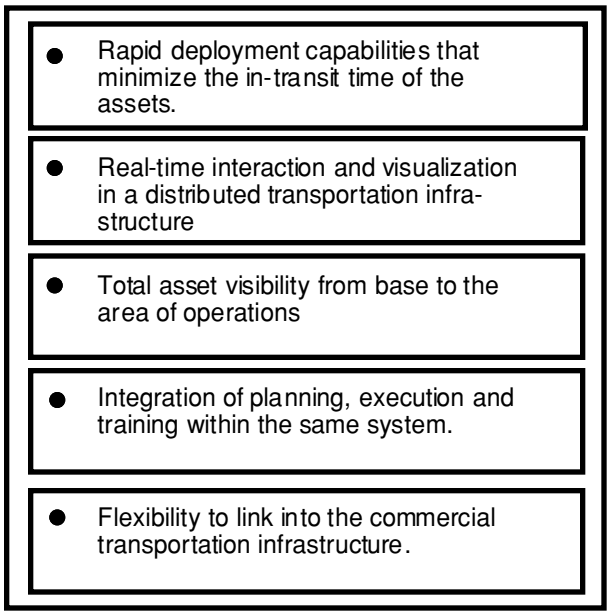


Figure 51: Military Deployment Objectives

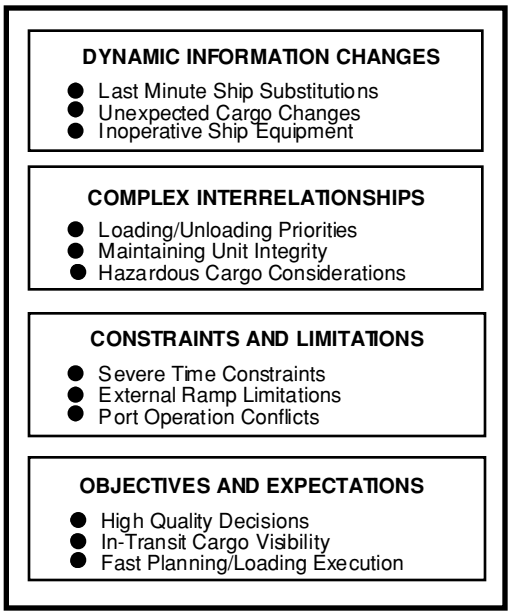


Figure 52: Stow-Planning as a Complex Problem

The CADRC was asked to focus on the stow-planning operations that occur at the POE. As shown in Figure 52, ship load-planning has many of the characteristics of a complex problem situation. First, there are continuous information changes. The vessel that arrives at the port may not be the vessel that was expected and that had been planned for. This means that the existing stow-plan is no longer applicable and a new plan has to be developed. Similarly, last minute cargo changes or inoperative lifting equipment (e.g., cranes) may require the existing plan to be modified or completely revised.

Second, there are several complex interrelationships. The cargo on any one ship may be destined for several ports of debarkation (PODs), requiring careful consideration of loading and unloading sequences. However, these sequences must take into account unloading priorities that may be dictated largely by tactical mission plans. In addition, the placement of individual cargo items on-board the ship is subject to hazardous material regulations and practices. These regulations are voluminous, and complex in themselves. At times they are subject to interpretation, based on past experience and detailed knowledge of maritime risks and practices.

Third, there are many constraints and limitations. Some of these constraints are static and others are dynamic in nature. For example, depending on the regional location of the port external ship ramps may not be operable under certain tide conditions. Local traffic conditions, such as peak hour commuter traffic and rail crossings, may seriously impact

the movement of cargo into staging areas or from staging areas to the pier. While these constraints are compounded whenever loading operations occur concurrently, the general complexity of the stow-planning problem is exacerbated by the number of parties involved. Each of these parties plays an important role in the success of the operation, but may have quite different objectives. Certainly, the objectives of the Union-governed commercial stevedore crews that carry out the actual loading tasks are likely to differ markedly from the prevailing military objectives (e.g., rapid loading and unloading operations, safety, unit integrity, load density, documentation accuracy, and security).

6.2 Objectives of the ICODES Prototype

The CADRC was asked to apply its ICDM model to the development of an integrated planning environment in which cooperating agents continuously assist human operators in the ship load-planning activity. Specifically, it was proposed to develop a proof-of-concept system that would demonstrate the feasibility and nature of a tool that could effectively assist ship stow-planners by providing timely advice based on expert knowledge and actual experience (i.e., lessons learned). The resulting proof-of-concept Integrated Computerized Deployment System (ICODES) was developed in five months to the following specified performance characteristics:

- Increased productivity by providing greater speed and/or the ability to develop and evaluate alternative solutions. Specifically, it was stipulated that ICODES should be able to support the load planning of up to four ships concurrently, in the same session.
- Higher quality plans by predicting problems and preventing their occurrence.
- Improved responsiveness to unplanned changes and unforeseen contingencies.
- Facilitation of the cargo placement task during the stow-planning process by providing intelligent assistance and automatic options to the human ship stowage operator.
- Ability to serve as a training simulator for new ship stowage operators.
- Provision of a user-friendly, graphic (multi-media) user-interface.

6.3 System Description and Architecture

The ICODES proof-of-concept system consists of a centralized data-blackboard, a CAD (Computer-Aided Drawing) engine, two user-interface modules, 10 rule-based agents, and a multi-media (video) facility. The implementation design emphasizes local decision-making, distributed processing, collaborative problem solving, user-computer interaction, computer-based assistance, visualization through multi-media capabilities, connectivity,

and integration. The agents interact with the data-blackboard by sending and receiving information to and from a frame-based semantic network, maintained by the blackboard (Figure 53). All of the rule-based components are written in the CLIPS (NASA 1992) expert system shell language, and the procedural components are written in the C++ language (Stroustrup 1987). The data-blackboard, main user-interface, and message passing facility are compiled together into a single process with CLIPS as an embedded component (Figure 54).

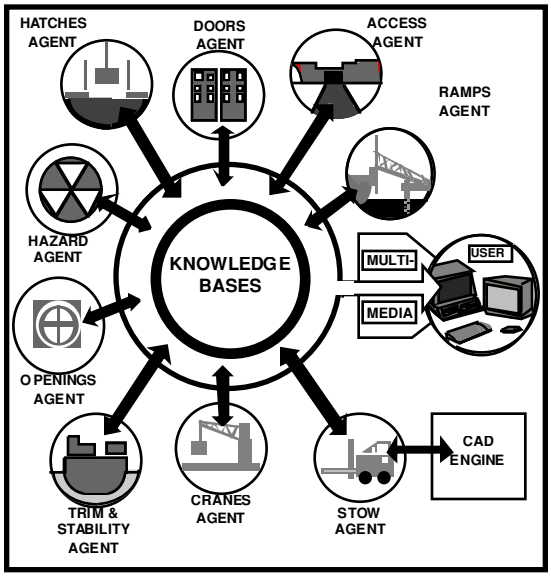


Figure 53: ICODES Implementation Design

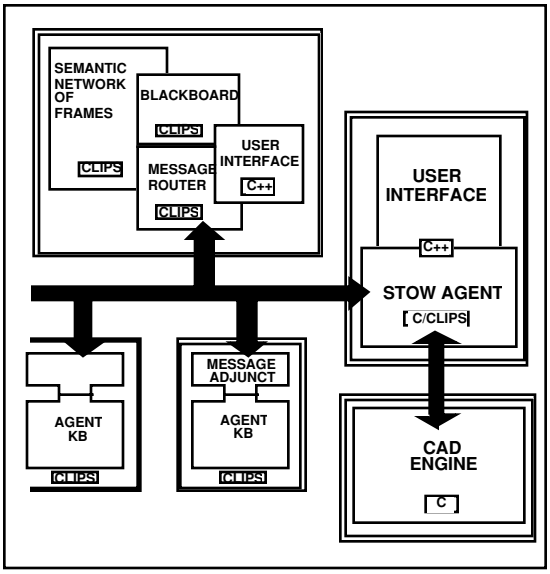


Figure 54: ICODES System Architecture

AutoCAD (Autodesk 1993) was selected to provide the drawing facilities required of the proof-of-concept system, primarily in order to easily incorporate the existing cargo and ship diagrams. ZINC (Zinc 1993), a graphic user-interface development package, was selected to provide rapid design of dialogue and menu presentation and processing, in particular since it can produce both X-Window (Scheifler and Gettys 1992) and Motif (Gregory 1992) code from the same source. Further, the RogueWave class library for C++ (Rogue Wave 1992) was selected to provide the platform for working with the C++ code produced by ZINC; and to also provide efficient object level programming support for the interface coding itself.

These software packages can support all of the individual requirements for a modern intuitive interactive graphic user-interface. Unfortunately, their integration within a single process is a complex task. For example, the X-Window code used to support Motif utilizes an event queue to monitor all X-Window events. However, the ZINC package provides another event queue for its own events. In order to obtain the proper actions with X-Window and ZINC events it often became necessary to program the posting of events in the second event queue when processing an event in the first event queue.

AutoCAD is less cooperative. In order to request AutoCAD commands from within the programming environment, as opposed to the AutoCAD interactive user command

environment, it is necessary to make C function calls available through the AutoCAD Development System (ADS). However, the ADS functions do not naturally support all of the user command facilities. In addition, there is a high degree of overhead since ADS creates calls to LISP functions that are native to AutoCAD execution. More importantly, the ADS calls must be limited to discrete activities because AutoCAD is not receptive to interruption while it is engaged in the execution of an ADS function. This means that other concurrent actions within the ICODES system as a whole are stalled until an ADS function has finished executing.

A greater problem is associated with the realization of a high degree of concurrence in communication among the various components. The socket code available from earlier ICDM models, for operation with a data-blackboard and general agents, works in a very special way with X-Window. Ordinarily in an X-Window environment when there are no current events the X-Window server becomes idle, waiting for an event. In the ICDM models it may be necessary for an agent to process the reception of a message, while it is in this idle X-Window event state. This can be achieved by requiring X-Window to also monitor socket events. Under these conditions X-Window will execute a socket message handler when a message is received by an agent (i.e., X-Window waits for both socket events and X-Window events). However, in the case of a CLIPS agent operating within the ICDM framework the message handler activity is confined simply to the assertion of facts into the fact-list of the agent. The handler does not cause the agent code to execute. However, within the context of the ICDM model it is important that the agent executes when a message is received, in order to maximize the concurrence of communication within the system. The ordinary operation of X-Window requires the execution of the message handler code, without giving control to the agent. Instead, X-Window waits for an event. While it is possible to cause X-Window to execute the agent code, the mechanism for doing this is not compatible with the manner in which ZINC operates. Thus it became necessary to assert a kind of 'dummy' X-Window event from within the message handler and use this 'dummy' event to force X-Window to execute the agent as soon as possible after it receives a message.

Several other changes were made in the ICDM data-blackboard and its communication mechanism. Some minor changes permitted new relationships and frame representations. A major change was the decision to maintain all of the semantic network information in a single CLIPS fact-list. Thus the conceptual notion of the semantic network being stored in the data-blackboard is fully realized in the ICODES system.

In previous ICDM model implementations the semantic network facts were distributed to the agents, but not necessarily retained in any one place. Even though the semantic network is available in one CLIPS fact-list, its information is not immediately accessible to the user-interface facilities. Yet the user-interface must be capable of producing reports, such as lists of cargo items, from the information held within the semantic network. In order to realize this need a special fact 'query' facility was developed in C++ (Stroustrup 1986). This facility allows facts that describe the report information to be asserted from the C++ to the CLIPS portions of the user-interface. The assertion from the C++ environment causes a single rule to fire in the CLIPS environment, which also

contains the semantic network. The protocol involved guarantees that the rule executed will invoke a user defined function that will in turn search the CLIPS fact-list for the required information, and then build a RogueWave list object of the results. The list object holding the results is then available to the C++ environment when it resumes execution. This new facility permits hundreds of items to be retrieved from the semantic network in a very short period of time.

The basic ICODES architecture, shown in Figure 55, features four principal processes as follows: main user-interface; CAD interface; agents, executing within CLIPS processes; and, X-Window system. The main user-interface couples the ZINC produced code, additional C++ interface code, a CLIPS system (i.e., 'uiclips' with an associated knowledge base), and C code (referred to as the "communications adjunct" in Figure 55). A few minor source code changes, user function differences, and knowledge base rule variations exist in the different CLIPS processes identified as 'uiclips', 'cadclips', 'tsclips', and 'agentclips'. Not only is it desirable, based on the distributed cooperative computing philosophy, to have separate CLIPS processes but it is also efficient to customize the various CLIPS executables for the tasks that they will perform. The 'uiclips' process is quite unique in that it supports the data-blackboard, containing the semantic network. It must therefore house the message router rules that collect the 'input templates' from the agents connected to the blackboard and then distribute the semantic network items as requested in the input template of each agent.

The communication between the processes is realized through UNIX sockets. The code that supports the socket functions is written in the 'communications adjunct'. Since this code is written in the C language, it was compiled with the C language source code that makes up 'uiclips' and the C++ code that comprises the user-interface. Thus the entire main user-interface can be viewed as C++ code (Figure 54).

The drawing interface consists of AutoCAD and its special set of ADS functions, more ZINC and C++ interface code, the 'communications adjunct', and another minor modification of CLIPS called 'cadclips'. This process is primarily concerned with the presentation of graphic icons to represent ship and cargo items, and to generate frames that provide the real world attributes relating to actions represented by the drawing activities. For example, when a cargo item is placed into a ship compartment, this process must send the appropriate facts, identifying the cargo item and its location, to the semantic network. The same process also monitors drawing related features such as the intersection of cargo items with one another, the vessel, and the environment external to the vessel. In both the main user-interface and the CAD interface processes there are direct Motif calls for X-Window actions, primarily created through ZINC. This is represented by the clear X-Window oval box in Figure 55.

The darker X-Window oval box represents the X-Window processing that is associated with the 'dummy' events mentioned earlier. This processing is required to implement the concurrent monitoring of messages received through sockets in the various processes, and serves as a reminder that the communication facilities are related to the X-Window system.

The other processes in ICODES are used to implement agents. Most of the agents use the same version of CLIPS, called 'agentclips'. An exception is made for the Trim and Stability agent, which requires some special user functions. Also, it must be mentioned that the Assisted-Stow agent does not exist as an independent process. Since Assisted-Stow needs to work intimately with the drawing facilities, it is more efficient to load its rules within the 'cadclips' knowledge base, which houses the User-Stow rules as well. The resulting architecture provides an efficient implementation of the required components of the ICODES proof-of-concept system.

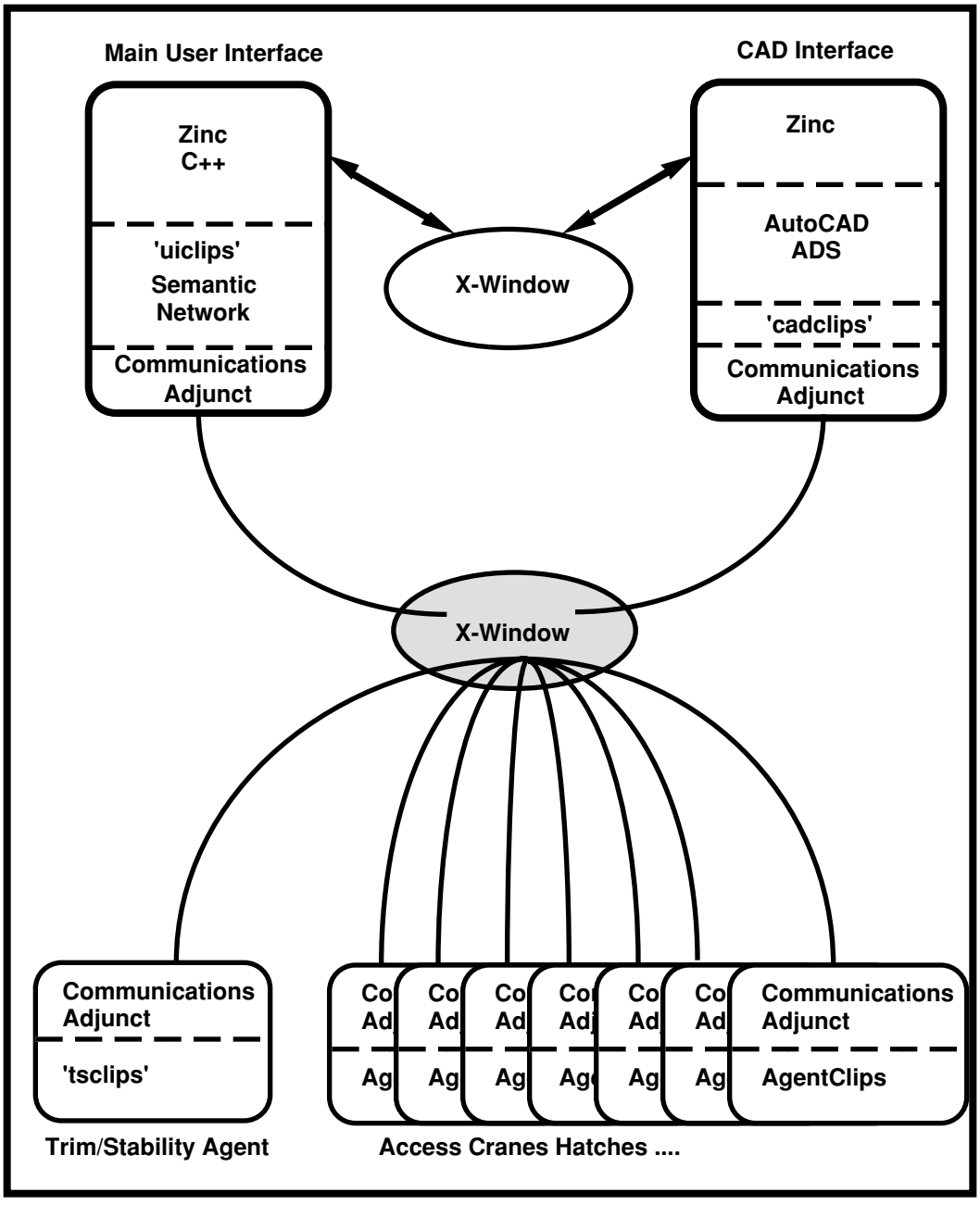


Figure 55: ICODES Prototype System Architecture

6.3.1 Data-Blackboard and Agents Interface

The data-blackboard is a CLIPS expert system that has several responsibilities. First, it serves as the repository of a semantic network of frames. In order to accomplish this function the data-blackboard also includes communication facilities to all other processes that require access to the semantic network. There are several methods, such as shared memory, named pipes, and sockets, available for accommodating these communication requirements. Since the socket code for this purpose was available for use from existing mature ICDM implementations, it was employed in the ICODES prototype system.

Second, the data-blackboard incorporates a message router to distribute the semantic network items to agents and in general to any processes that request a connection to it. Each fact that is used in the frame representation of a semantic network item can be sent via a socket to any process that is connected to the blackboard. Agents that are CLIPS expert systems will be interrupted by the reception of such messages and the facts will be asserted into their fact-lists. Non-CLIPS expert systems will receive the facts as strings. Functions in C and C++ code can be called to translate the facts into the selected representation within the non-CLIPS component. Similarly, functions can be called in any of the component environments to transmit a string-encoded fact to the data-blackboard. After receiving a message, the data-blackboard decodes the string and asserts the CLIPS fact into its fact-list.

As discussed earlier, the ICODES architecture consists of multiple processes interacting with one another via inter-process messages. Physically, each agent within the system is connected with the data-blackboard via Internet sockets. The use of Internet sockets allows for site-independent communication. Each agent monitors its own blackboard connection for incoming communication. Similarly, the blackboard monitors each of its agent connections for incoming messages. In all cases these incoming communications can be processed immediately or buffered for processing at a later time.

The CLIPS process (i.e., 'tsclips' for the Trim and Stability agent, 'cadclips' for the Stow agent, and 'agentclips' for all other agents) under which each agent executes consists of the fact management and inference engine capabilities provided by CLIPS in addition to a TCP/IP-based communication adjunct. The adjunct essentially provides its agent two-way communication with the data-blackboard. Agents send messages to the blackboard by issuing a call to a special 'assert' function added to the CLIPS shell. The syntax and semantics of this call parallel the normal CLIPS assertion protocol with the exception that the associated fact is asserted directly into the fact-list of the blackboard rather than the fact-list of the agent.

Facts from the blackboard are received by agents in basically two ways. The first method is through an explicit reception within a CLIPS rule, and the second method relies on reception between rule-firings. In the latter case, at the completion of each rule-firing the communication adjunct is queried for any pending messages. If a message exists, it is processed immediately. Processing of all incoming messages takes the form of transparent assertions to the receiver's fact-list. The receiver is notified of an incoming message by having an appropriate rule placed on, or removed from, the current agenda.

6.3.2 Semantic Network

In order for the expert systems (agents) to exchange data, they must share a common representation of information (i.e., data and relationships). A frame-based representation was used to represent objects and their attributes in the ICODES proof-of-concept system. A frame is a collection of information about an object or a class of objects.

In the ICODES frame management system three fact types define the different parts of the object data; namely, FRAME, VALUE, and RELATION. The FRAME fact provides the class name and instance identifier that links the various attributes of an object together.

(FRAME <class> <instance> <source> <type>)

where: FRAME is a keyword;
<class> is the name of the class of this frame;
<instance> is the frame identifier;
<source> identifies the origin of the fact;
<type> indicates the type of fact (i.e., whether it is a 'suggested' or 'current' value).

The VALUE fact contains information about a particular attribute or slot of a FRAME class.

(VALUE <class> <instance> <attribute> <value1> [<value2> <value3>] <source> <type>)

where: VALUE is a keyword;
<class> is the name of the class of this frame;
<instance> is the frame identifier;
<attribute> is the slot name or attribute;
<value1> <value2> <value3> contain the actual value of this slot;
<source> identifies the origin of the fact;
<type> indicates the type of fact.

The RELATION fact is used to connect one FRAME with another FRAME.

(RELATION <relation-name> <class1> <instance1> <class2> <instance2> [<class3> <instance3>] <source> <type>)

where: RELATION is a keyword;
<relation-name> defines how the two FRAMEs are related;
<class1> <class2> <class3> are class identifiers;
<instance1> <instance2> <instance3> are instances of <class1> <class2> and <class3>;

<source> identifies the origin of the fact;
<type> indicates the type of fact.

In the following example the first fact (FRAME) specifies that there is a frame for the class "vessel" which has "111" as its identifier. This fact was generated from the MARR file and is currently valid. The second fact (VALUE) states that the name of the vessel "111" is "Maersk Constellation". The third fact (FRAME) specifies that there is a frame for the class "deck" which has "222" as its identifier. The fourth fact (RELATION) indicates that vessel "111" is related to deck "222" and that a "has_a" relationship exists between the two frames. In other words, vessel "111" has a deck with an identification code of "222".

(FRAME vessel 111 MARR current)
(VALUE vessel 111 shipname "Maersk Constellation" MARR current)
(FRAME deck 222 MARR current)
(RELATION has_a vessel 111 deck 222 MARR current)

6.3.3 Main User Interface

This Section describes the main user-interface component that provides user access to all options other than the interactions with the AutoCAD drawing environment (Figure 56). The separation of the CAD user-interface was dictated by representational requirements, which are discussed later in Section 6.3.4.

To accommodate the fairly wide range of requirements and differing operational preferences of stow-planners, the user-interface was designed to be flexible to any sequence of events, and display information in a user-customizable format. The programming philosophy in which the user guides the system, and not vice versa, is often referred to as 'user-driven'. In a user-driven environment the system does not dictate a sequence of events to the user. Rather, all options are available to the user at all times, and the system largely responds to actions that the user initiates.

To realize these requirements, the user-interface must have dynamic access to all information in the system at any given time. This requires an efficient method of retrieving large amounts of information from the semantic network. These considerations led to two major implementation design decisions for the ICODES proof-of-concept system. First, it was agreed that the blackboard should keep a local copy of the semantic network. This provides a single location where all current information can be found at any given time. Second, it was decided that the user-interface and the blackboard should be compiled together as a single process. This allows the user-interface to gain direct access to the entire semantic network without the overhead or complexity of inter-process communication facilities.

While combining the blackboard and user-interface into one process solves several performance problems, it introduces several others complications. The blackboard is written in CLIPS, which is appropriate given the pattern-matching requirements of

interacting with other system components. However, the user-interface, being a largely procedural program, requires a more procedurally-orientated language such as C++. This required the combination of CLIPS and C++ in the same process. It also required that the combined blackboard and user-interface process be simultaneously receptive to the user through the user-interface, and to messages from other agents through the blackboard.

When information is required in the user-interface a special C++ function called 'runQuery' allows information of any type to be directly extracted from the CLIPS fact-list. The 'runQuery' function accepts queries, which are dynamically built in the user-interface from user actions. Upon completion of the 'runQuery' function, a list of the specified information is returned to the user-interface and displayed for the user. This method of extracting information from the semantic network through the C++ component of the combined blackboard and user-interface process is used by every functional element of the user-interface that requires information from the semantic network.

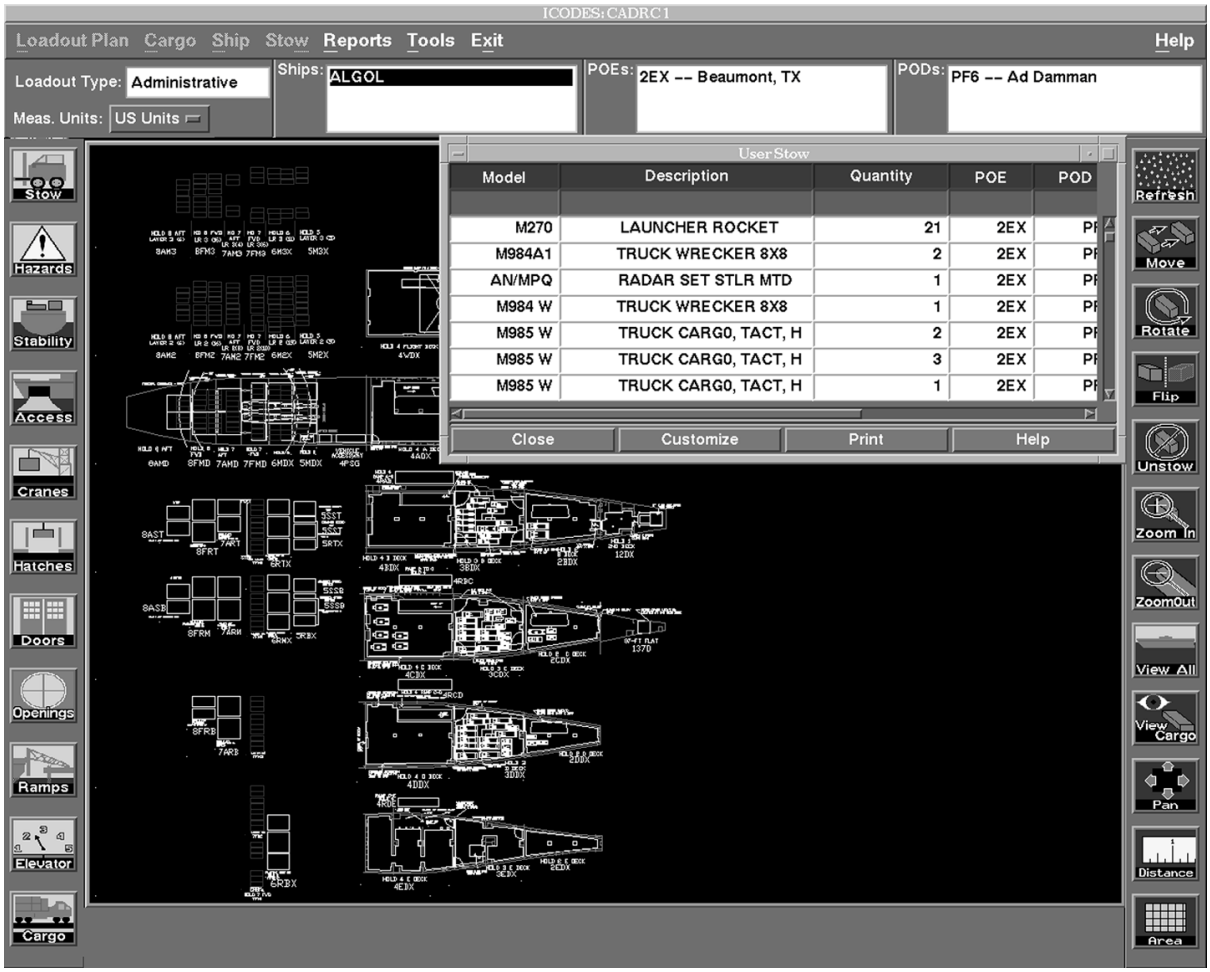


Figure 56: Main Screen of the ICODES Prototype Application

6.3.4 CAD User Interface and Stow Agent

The ability to embed intelligence in a computer-based decision-support system is largely predicated on the existence in the system of a level of knowledge representation that is similar to that employed by human decision-makers. In the stow-planning application this requires that the system be capable of maintaining high level representations such as vessel, compartment, crane, ramp, and cargo item. Existing commercially available CAD systems, such as AutoCAD, do not understand such real world objects but are limited to the recognition of lower level geometric objects such as points, lines and polygons.

One approach to solving this fundamental problem is to link the CAD system to a front-end process that maintains the high level representation, through interaction with the user and internal reasoning capabilities, and uses the CAD system simply as a drawing engine (Myers et al.1993). Under these conditions the front-end process completely controls the operation of the CAD system, sending instructions and receiving the results of the execution of these instructions as the system or user environment dictates. AutoCAD provides a programming interface (ADS) that can be used to drive its drawing functions from an external process. This facility was used in the ICODES proof-of-concept system, together with CLIPS and ZINC, for the manual (User-Stow) and automated (Assisted-Stow) cargo placement functions. Accordingly, the CAD interface consists of ADS, CLIPS and ZINC components that are combined into a single process. This process is referred to as the Stow agent.

Interactions with AutoCAD comprise three distinct stages of execution: the CLIPS stage; the 'userfunctions' stage; and, the ADS stage. During the CLIPS stage the Stow agent receives facts from the blackboard or its own CAD user-interface component, calls CLIPS userfunctions, and then reasserts facts to the blackboard. All of the rules that handle these operations are located in the 'stow.kb' knowledge base. The facts sent from the blackboard to the Stow agent use the key word 'CAD' to distinguish them from blackboard communications to other agents. This allows the message router in the blackboard to automatically send these facts to the Stow agent. The 'userfunctions' stage consists of calling ADS functions and formulating facts that are then asserted into the Stow agent's fact-list. The ADS stage handles all calls to ADS and API functions, takes in all graphical input to the CAD system and typically returns a structure to the user-function that called it.

If an action is initiated from the CAD user-interface, then facts are asserted into CLIPS from the Stow agent process itself rather than from the blackboard. Each time an item in the CAD user-interface is selected a corresponding function is called which asserts a fact into the Stow agent's fact-list. One of the few exceptions to this typical sequence of events occurs in the case of the Assisted-Stow option. Since the Assisted-Stow option is an integral part of the Stow agent process, the rules that deal with the automatic templating of cargo items are driven by local facts rather than those that come from the blackboard. This eliminates the message passing overhead, thereby improving the performance of the automatic templating function.

6.3.5 The Accessibility Agents

The family of 'accessibility' agents is concerned with determining and reporting on the ability of cargo items to access any ship compartment. The functionality of performing this analysis task has been divided into several domains that are represented by the following agents: Access; Cranes; Doors; Hatches; Ramps; and, Openings. Although these agents are functionally related and cooperate closely with each other to analyze the accessibility problem, they are nevertheless separate processes (Figure 57).

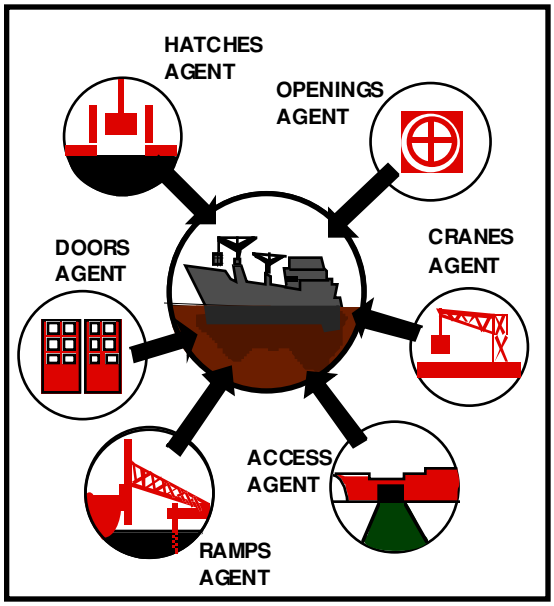


Figure 57: The Accessibility Agents

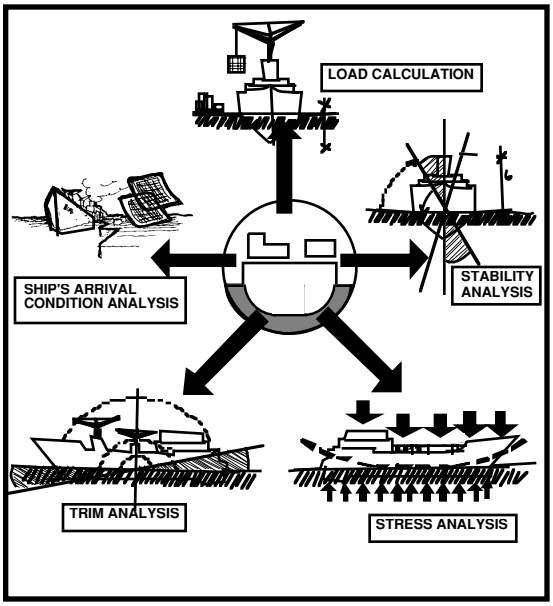


Figure 58: Trim and Stability Agent Functions

The Access agent assigns the access permissions based on information provided by the other 'accessibility' agents. If any of that information changes (e.g., if the dimensions of a cargo item are updated at run time), the access permission is reexamined. The Access agent also indicates a violation when an item is placed into a compartment and access permission is not set.

The Cranes, Doors, Hatches, Ramps, and Openings agents each examine the accessibility of a cargo item into a compartment based strictly on the particular concerns of its domain. For example, the Hatches agent determines only whether a cargo item is able to fit through a hatch. It does not concern itself with the ability of a crane to lift the item into position (the Access agent handles this coordination). Since cranes (or booms) may be married together to produce the capability of lifting heavier items, the Cranes agent is also responsible for determining which booms may be married together, what holds the married booms can reach, and their combined weight limit. The Cranes agent determines all of the possible marriages and related information during system initialization. In addition, the Cranes agent examines the following crane characteristics to determine accessibility:

- the status of each boom;

- the weight capacities of booms and cargo items, respectively;
- the holds that can be reached by each boom.

The Doors agent examines the following characteristics relating to doors and cargo items:

- the status of each door;
- the widths of doors and cargo items, respectively;
- the heights of doors and cargo items, respectively.

The Hatches agent examines the following characteristics relating to hatches and cargo items:

- the status of each hatch;
- the widths of hatches and cargo items, respectively;
- the lengths of hatches and cargo items, respectively.

The Ramps agent examines the following characteristics relating to external ramps and cargo items:

- the status of the ramp;
- the widths of the ramp and cargo items, respectively;
- the lengths of the ramp and cargo items, respectively;
- the capacity of the ramp and the weights of cargo items;
- the clearance of the ramp and the heights of cargo items.

The Openings agent examines the following characteristics relating to openings and cargo items:

- the status of each opening;
- the widths of openings and cargo items, respectively;
- the lengths of openings and cargo items, respectively;
- the heights of openings and cargo items, respectively.

In the ICODES proof-of-concept system the 'accessibility' agents determine the access permissions for all cargo items (i.e., whether a cargo item can be moved or lifted into any compartment) as early as possible. The majority of these calculations are performed during the system initialization stage, immediately after the cargo list has been loaded into the semantic network. However, the addition of any new cargo items during execution (such as items produced by a marriage) will automatically activate the 'accessibility' agents to determine the full set of access permissions for the new cargo item.

Both the Access and Crane agents have reporting capabilities that are invoked by clicking onto the status window of the particular agent. The Access agent report informs the user whether a particular cargo model has access to a given compartment. It includes a list of findings from each of the other 'accessibility' agents, to indicate precisely what access permissions are available. The Crane agent report provides a list of booms that are capable of lifting a particular cargo model into a given compartment (as identified by hold and deck). Instead of listing all possible boom combinations, it recommends what it

considers to be the best boom marriages. For example, if a cargo model requires a two-boom marriage, only two-boom marriages are listed (i.e., other possible marriages involving more than two booms are not listed).

6.3.6 The Trim and Stability Agent

The Trim and Stability agent is capable of determining the 'arrival condition' of the vessel, and continuously monitors the vessel during the stow-planning session by performing load calculations, trim and stability analyses, and stress calculations (Figure 58). When the Trim and Stability agent receives information that identifies the vessel from the data-blackboard, it looks up trim and stability tables to obtain values for the required set of variables, such as the ship's metacentric height (GM) requirement and the various vertical center of gravity (VCG) values under different loading (weight) conditions. As the agent receives information about the placement of cargo items, it calculates the cargo weight and the VCG of the hold. It also considers each of the ship's tanks and calculates the tank's weight and VCG. Again, the trim and stability tables are used in these calculations.

At appropriate times, or when requested by the user, the agent posts the following primary stability results on the semantic network:

- the ship's total weight (W);
- the ship's center of gravity from the keel (KG);
- the available GM;
- the GM required.

Trim calculations are performed in a similar manner. Additional information about the vessel is obtained from the trim and stability tables. This includes the light ship longitudinal center of gravity - forward perpendicular (LCG-FP), the crew and store LCG-FP, and the mean salt water draft. Using the information gained from the data-blackboard about the cargo in each hold, the Trim and Stability agent uses the trim and stability tables to calculate each hold's LCG-FP. For each of the ship's tanks it also uses type and weight information from the data-blackboard to determine the tank's LCG-FP. The results of these calculations are compared to the stability requirements of the ship and the user is alerted if a safety violation is detected.

6.3.7 The Stow Agent

The Stow agent is capable of operating in two modes (Figure 59): as a manual cargo placement tool (i.e., User-Stow); and, as an automatic prorating and templating facility (i.e., Assisted-Stow). In the User-Stow mode the stow-planner selects a cargo item from a textual cargo list and graphically drags it into the desired ship compartment displayed in the CAD window. As soon as the cargo item has been positioned the agents, each in its particular domain, analyze the cargo location in respect to its trim and stability impact, accessibility, hazard infractions, and other placement violations. Functions are available to allow the user to 'move' or 'unstow' the placed cargo item. During the placement task

interference checking is performed by AutoCAD and the results are communicated via a front-end ADS process to the data-blackboard, where they are posted on the semantic network.

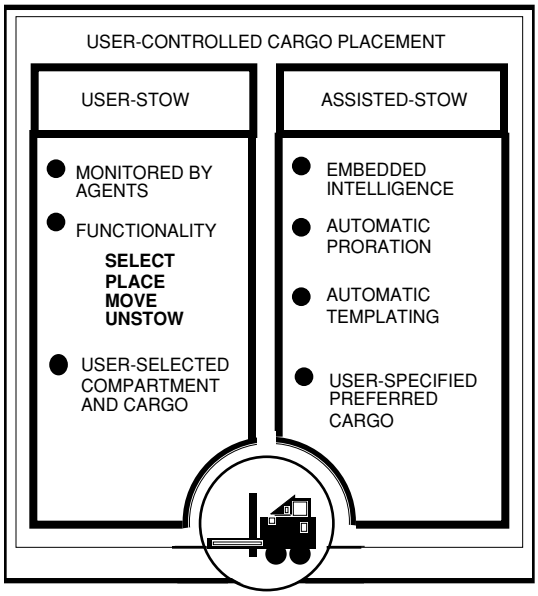


Figure 59: Stow Agent Functions

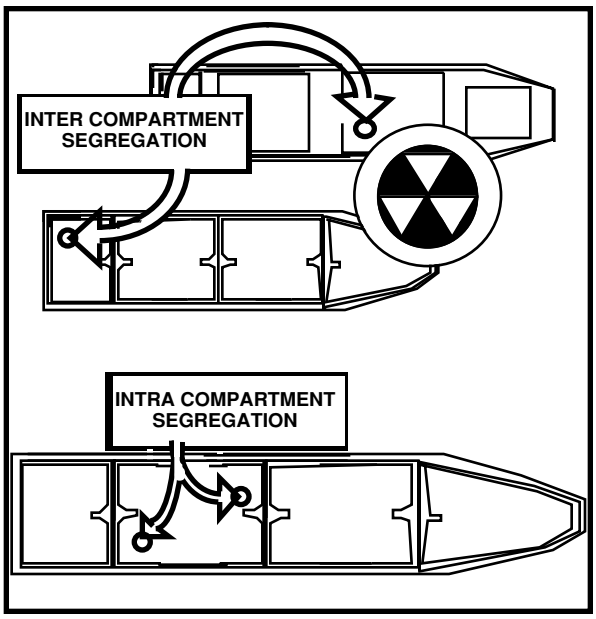


Figure 60: Hazard Agent Functions

In Assisted-Stow mode the user specifies the decks and/or holds to be loaded, the target area utilization percentage (i.e., percentage of each compartment to be filled with cargo, - often referred to as the 'stow factor'), and any preferences or restrictions that should be applied to the cargo. The Stow agent then proceeds to analyze the entire cargo list and all ship compartments to determine its selection of cargo for the specified compartments.

By the time the Assisted-Stow operation is invoked, the Access agent has already determined all compartments to which each model version has access. The type of access is also known, that is, whether items can be rolled on/off, or if they must be lifted in position. The prorating process then proceeds in four phases:

Phase 1: Preferred item placement.
Phase 2: Priority item placement.
Phase 3: Weight distribution placement.
Phase 4: Automatic templating.

During the *'preferred item placement'* phase the Stow agent identifies all cargo items that have been specified for preferred placement by the stow-planner. These items are placed first. In phase 2, compartment(s) are selected for each item by consulting a priority fact-list that lists the most desirable compartments for a particular model version. For example, it may be generally preferred to tow rather than lift helicopters onto a ship, hence the Stow agent will first select compartments on the RORO (i.e., roll-on roll-off) deck. If there is no specific priority list for the cargo item or model version under consideration then the Stow agent will consult a default list of compartments. The agent

will attempt to place the cargo item in the first compartment selected. It checks to see if the item has access, and if there is sufficient room (square footage) available in the compartment. If these conditions are met, the item is assigned to the compartment, and the total available square footage of the compartment is reduced by the cargo item's footprint area. This continues until all preferred items have either been assigned to compartments or cannot be assigned to any compartment.

In the *'priority item placement'* phase cargo items are selected based on a priority ordering, which takes into account the difficulty and/or sensitivity of each cargo item. The ranking logic is based on advice received from expert stow-planners and includes the following in descending order of restrictiveness:

1. hazardous material (excluding explosives);
2. sensitive cargo (e.g., weapons);
3. prioritized cargo that must be unloaded first (e.g., wreckers);
4. breakbulk (underdeck stow only);
5. breakbulk (oversized/on deck stow only);
6. lift-on only cargo (e.g., bridge sections);
7. helicopters and aircraft;
8. deadlined vehicles (i.e., vehicle cannot be moved);
9. deadlined vehicles (i.e., vehicle can be towed);
10. container without chassis;
11. steel tracked vehicles;
12. semitrailers with large kingpins;
13. semitrailers with standard kingpins;
14. oversized wheeled vehicles;
15. large wheeled vehicles with trailers;
16. large wheeled vehicles without trailers;
17. tracked vehicles;
18. light wheeled vehicles with trailers;
19. light wheeled vehicles without trailers.

In the *'weight distribution placement'* phase the remaining cargo is placed so as to achieve a satisfactory weight distribution. Cargo items are selected by weight, from heaviest to lightest, and compartments are selected from the lowest to the highest deck. If more than two-thirds of the total cargo weight has been placed on the bottom deck, no more items are placed there. At the end of this placement phase, all cargo has either been assigned to a

compartment, or cannot be placed in any available compartment. Cargo that cannot be placed is specified as CLOP (Cargo Left On Pier).

During the *'automatic templating'* phase the Stow agent determines the actual placement and coordinates for items within a particular compartment. Since the placement of cargo items during the previous three phases has been subject to available area, on a square footage basis, it is possible that during templating some items may not fit into a compartment. Such items are then reclassified as CLOP. As the cargo item's coordinates are determined, the items are placed into the ship drawing.

6.3.8 The Hazard Agent

The Hazard agent continuously monitors the placement of cargo items containing hazardous material. It determines the validity of cargo placement relationships with other hazardous cargo items. If the Hazard agent concludes that a violation has occurred it alerts the user by turning the surround of its status window red. The knowledge embedded in the Hazard agent was derived from interviews with experienced stow-planners and the Bureau of Explosives (BOE-6000-L) manual (Bureau of Explosives 1992). The types of placement violations that the Hazard agent is capable of detecting (Figure 60) include:

- segregation incompatibilities among hazardous cargo classes;
- spacing requirements and restrictions;
- legal locations on board a vessel.

The Hazard agent operates in a reactive mode, responding to the cargo that has been stowed on the ship. For example, if a breakbulk cargo item carrying acetylene (i.e., a flammable gas) has been stowed in a compartment, and another cargo item containing batteries (i.e., a corrosive chemical) is placed into the same compartment the Hazard agent will check the distance between the two potentially hazardous cargo items to determine if the separation requirements have been violated. If yes, then the agent's status window alerts the user by turning red. The user may obtain a report with a detailed explanation of the violation by clicking onto the agent's status window. In this case the Hazard agent will indicate that a minimum separation distance of 20ft (6m) is mandated.

The Hazard agent is able to distinguish among similar regulations pertaining to different locations (i.e., compartments) on the vessel. If the above example had occurred on the weather-deck, which is exposed to the open air, then the hazard agent would take into consideration that the segregation rules are more lenient. Furthermore, if the cargo items carrying the hazardous material are classified as "transports" (e.g., trucks), then the Hazard agent would also take into account the "container relationship". It recognizes both "open" (e.g., open truck bed) and "closed" (e.g., truck bed with closed cover) container relationships, and understands the distance requirements that pertain to these conditions. The Hazard agent is also able to analyze situations involving containers, such as CONEX boxes and MILVANs, which contain hazardous material. These situations are computed in a similar manner to the "transports" classification.

If a violation occurs the Hazard agent provides a report that describes the type of violation and indicates:

- the locations of the incompatible hazardous cargo items;
- the severity of the violation;
- the determining factors of the violation;
- special provisions pertaining to any of the hazardous materials.

For instance, in the previous example involving a cargo item containing acetylene, the Hazard agent's report would list three *"other-provision"* codes as follows:

25 Shade from radiant heat;
40 Stow "clear of living quarters";
57 Stow "separated from" chlorine.

6.4 The ICODES End-User Application

Following the completion of the proof-of-concept system described in this Section the CADRC entered into a three-year contract for the development and fielding of an end-user product. The ICODES application was fielded for 'beta' testing in January 1997 and is scheduled to be installed at MTMC ports during 1997/98. In the final product the agents were recoded in the C++ language to maximize performance, and the ZINC graphic user-interface development tool was replaced to avoid the potentially disruptive influence of its event queue on internal communications.

(Editors Note: Subsequently in 1999, ICODES Version 3 was fielded by the Military Traffic Management Command (MTMC) to the US Army. This initial ICODES product operated in a UNIX operating system environment on IBM Laptop computers and Hewlett-Packard workstations. During 2001 ICODES Version 3 was converted to execute under the Microsoft Windows NT operating system environment on Personal Computer (PC) workstations and Laptop computers. Also, during 2001 ICODES Version 4 was extended to accommodate the load-planning requirements of the Marine Corps and the Navy. Subsequently ICODES Version 5.2 was fielded to the Marine Corps at the end of 2002. ICODES Versions 5.1.6 and 5.2 were used as the system of record for the loading of virtually all ships used during the Iraqi deployment. Today, in 2003,ICODES Version 5.3 executes in a Microsoft Windows 2000 operating system environment on PC Laptop computers.)

7. FEAT: Military Planning with Multiple Service-Agent Systems

In 1994, the Collaborative Agent Design Research Center (CADRC) working jointly with CDM Technologies, Inc., undertook the development of a proof-of-concept implementation of the ICDM model in the military mission planning functional domain for the US Marine Corps (MARFORPAC). FEAT (Force Employment Analysis Tool), like other ICDM applications, emphasizes the use of agents, or modules of specific purpose and viewpoint, working in a collaborative manner within and, when appropriate, across separate computers. The agents are cast in a supportive role to the human decision-maker with the objective of monitoring user actions, presenting information in meaningful ways, and providing advice.

7.1 Military Planning and Execution as a Complex Problem

Much has been written in the literature about the complexities of planning and execution in warfare (Hayden 1995, Sawyer 1995, Schmitt 1994, Alexander 1993, Leonhard 1991, Creveld 1986, Clausewitz 1976). It therefore suffices here to summarize the salient characteristics (Figure 61). Military command and control is an information intensive activity, involving many variables with strong interrelationships. The information sources are typically widely distributed and subject to continuous change.

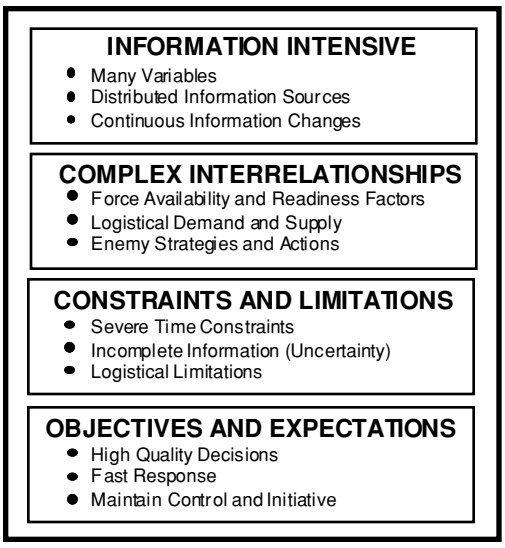


Figure 61: Nature of Military Planning

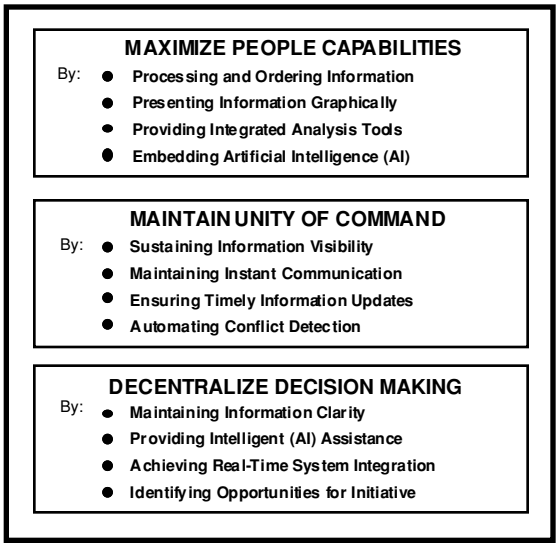


Figure 62: Military Planning System Objectives

Mission planning and execution are activities that are highly collaborative in nature, involving many players within a tightly knit team structure. The information needs of the individual team members are varied and often at least partially unpredictable. The reason for unpredictability is that the solution strategies adopted during the problem solving process determine to a large degree the kind of information required. For example, a battlefield maneuver that depends on air support will require significantly different information than one that utilizes naval fire support. Also, the information needs of the decision-makers are often poorly matched by the available data. In other words, although

the volume of available information may almost overwhelm the commander, the information may be unevenly distributed with a lack of useful data in some areas and an overabundance of data in other areas.

The relationships among the many variables, such as force availability and readiness factors, logistical demand and supply, and enemy strategies and actions, tend to be complex and subject to change. There is a high degree of uncertainty, not only in respect to the dynamically changing nature of the problem situation, but also in respect to the reliability and accuracy of the information itself. The constraints are also dynamic in nature. For example, a particular enemy action may fundamentally change the logistical supply capabilities of friendly forces. The time available to complete a planned maneuver may be drastically shortened and no longer time constrained following the occurrence of certain unforeseen events, or a sudden change in weather conditions may make it temporarily impossible to provide air support to ground forces that have been engaged by the enemy.

The objectives of a decision-support system, designed to assist human problem solvers confronted with the complexities of military planning, can be broadly divided into three categories (Figure 62). First, there is a compelling need to maximize the available human resources. To achieve this the system must be capable of processing, ordering, and appropriately filtering data into useful information. For this information to be communicated effectively to the users of the system and allow system components (i.e., agents) to assist in the decision-making activity, the information must be represented in the system in terms of objects. As discussed previously in Sections 3.2.2 and 5.2, a high level internal representation of the information that the decision-support system is processing is a prerequisite for embedding artificial intelligence components in the system.

Second, there is a need to maintain the unity of command. This can be accomplished through sustained information visibility, near real-time and fail-proof communication, and assistance in conflict detection. This assistance should be spontaneous and not dependent on user initiation. In other words, the system must continuously monitor the problem state to automatically detect actual and potential conflict conditions. Third, there is a need to decentralize the decision-making activity. This reduces the potential for communication bottlenecks and greatly increases the opportunities for parallel problem solving and the exercise of individual initiative.

7.2 System Description and Architecture

The FEAT prototype was designed to integrate into the existing military command and control systems environment as a knowledge-based decision support tool. It utilizes intelligent modules (agents) executing in parallel to determine the impact of decision alternatives and detect potential conflicts during the mission planning process. Remote FEAT workstations are able to collaborate in the planning activity using TCP/IP Internet protocol communication facilities. Principal FEAT capabilities include:

- real-time linkage to remote FEAT workstations and information sources;
- ability to send and receive messages;
- continuous analysis of planning state by multiple agents executing in parallel;
- automatic conflict detection;
- ability of user to plan concurrently within shared and private worlds;
- continuous access to 'Theater Backdrop' information.

ICDM applications emphasize the use of agents, or modules of specific purpose and viewpoint, working in a cooperative manner within and, when appropriate, across separate computers. Furthermore, ICDM models cast the agents in an assisting or supporting role to the human users of the system to effectively monitor user actions, present information in meaningful ways, and provide advice. The agents on each computer in an ICDM application react to information asserted to a common store referred to as a data-blackboard.

In FEAT a mechanism for collaborative planning is implemented to interactively link multiple users and data-blackboards, along with their agents. Users are able to work concurrently in two different environments, the ***private*** world and the ***shared*** world and send information from their terminal to another user's shared world. The recipients can then explore the reaction of the local FEAT agents to the new information, without interfering with any information in their private world. The objects that exist in the private and shared worlds are distinct. Further, the same agents operate in both worlds, but the results of agent action in the private and shared worlds may not be the same as though each world had its own set of independent agents.

The introduction of the private and shared worlds in FEAT makes it possible for the user to compare the implications of alternative decisions. FEAT also provides a third environment, the ***display*** world. Information that is placed into the display world is not seen by FEAT agents. However, it is available to all FEAT report and display facilities. Typically the display world is used to hold information that has come from a remote source. In particular it may be necessary to hold values from a remote terminal in the display world, because the action of the agents can create changes in the local objects received from another terminal. For example, one user might wish to receive information about a new activity, an exercise or operation, that has been planned at a remote terminal. In addition to the selection of units, dates, and other primary values, the local user may wish to see the cost data that was calculated at the remote terminal. This is easily accomplished by sending the cost data from the remote terminal to the display world of the local terminal. The primary data values are sent to the shared world of the local terminal and will cause the calculation of cost data to be made by the local agents. Then the local user can view the cost data calculated in the shared world by the local agents and compare that data with the cost data in the local display world (i.e., a copy of the information that came from the remote terminal).

The essential components of the system are shown in Figure 63 as a communication facility (i.e., CMS), several agents, a noteboard, a user-interface (i.e., display), a report module, and the ***shared***, ***private*** and ***display*** worlds.

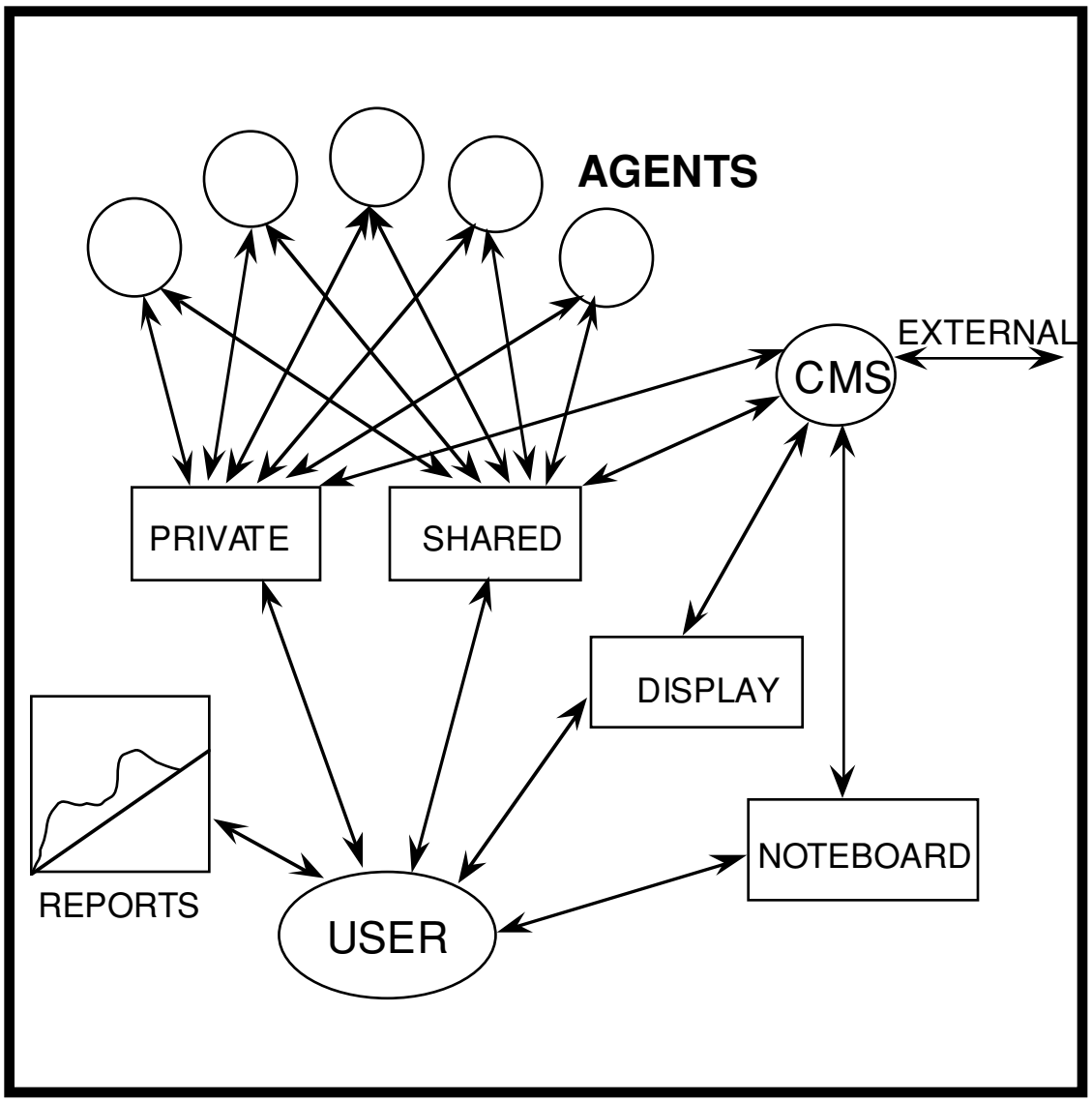


Figure 63: Conceptual View of the FEAT Architecture

7.2.1 The Semantic Network

The semantic network is the vocabulary of items that is made available to all components in the system. Of course any component may have its own local information and access specific files or databases. However it is intended that any information that is needed by any FEAT agent or report is included in the semantic network. The semantic network objects are implemented in two forms. The first implementation is defined in C++ classes. Considerable reliability and productivity leverage is provided through the use of the RogueWave class library (Rogue Wave 1992), particularly for use in the creation of lists of objects and operations such as sorting over those lists. All reports and interactions

with the user are implemented with C++ objects. The second implementation of semantic network objects is through COOL objects within the CLIPS expert system shell (NASA 1992). This implementation provides a corresponding object for inference and pattern matching functions for each object in the C++ version of the semantic network environment. In both the C++ and CLIPS worlds of objects, each conceptual semantic network class is instantiated as three separate objects, one object for each of the private, shared, and display environments. This permits the isolation and protection of the data that belong to the different worlds.

The graphics user-interface (GUI) and communication activities deal directly and efficiently with the C++ objects and the rule-based agents written in CLIPS work directly with their corresponding COOL objects. Any changes in the semantic network within one object representation (i.e., C++ or COOL) is immediately transmitted to the process that contains the corresponding object in the other form. Thus for each of the three worlds, ***private***, ***shared***, and ***display***, the C++ and CLIPS processes keep a one-to-one correspondence between their respective versions of the objects.

Object-oriented programming permits the updating of objects in the two different UNIX processes (i.e., C++ and CLIPS) to take place reliably by encoding the messages that initiate the update actions in each process within the methods that are used to accomplish the update within the other process. The advantage of object-level programming is underscored by the reliability achieved in updating the corresponding C++ and COOL objects. Every object in the semantic network is an instantiation of a class. The class is defined as a single entity and therefore the method, or function, that is used to set an attribute in every semantic network object instantiated from that class is the method that is defined in the single class. When an attribute in either a C++ or COOL object is set or modified, the communication of the update action to the other process is the method that was defined in the class from which the object was instantiated. The class method for performing the setting or modification of an attribute is the same method for every object instance of the class. Once the class methods are properly written the correspondence between the objects of the C++ and COOL worlds is guaranteed for all instantiations of the class.

7.2.2 The Blackboard Component

The internal FEAT coordination facility consists of the agents, the data-blackboard they use for communication, and the communication code used by the agents to correspond to the C++ world. The data-blackboard is important in at least two respects. First, it simplifies the lowest levels of communication. The user, as well as any of the rule-based agents, can communicate individually with any of the other agents. But more typically information is sent only to the one data-blackboard entity on a specific FEAT workstation (i.e., CPU). The data-blackboard then exports the information to all agents that have registered a request to import the information. Although requests to export and import information can be made dynamically, it is sufficient for agents to simply register requests for all objects of the types they handle when the agent code is loaded into the system. This permits communication from a sender to be directed to the collection of agents on any FEAT computer without having previous knowledge of the agents that are

available. New agents can be added to the system without any changes in the components that contribute (i.e., transmit) to the data-blackboard. Similarly, the agents can communicate (i.e., collaborate) with each other by posting values to the data-blackboard.

Second, the continuous activation of the agents connected to the data-blackboard eliminates all concern on the part of the user to identify what agent should be executing at any point in time. The agents respond opportunistically to the changes in the data-blackboard, activating rules as their conditions are met by the changes in the COOL objects and CLIPS facts asserted to the data-blackboard.

The CLIPS version of the data-blackboard is implemented as a 'Main' module in a single CLIPS process. This 'Main' module shares its objects and facts with the agent rule-bases that are implemented as secondary modules within the same CLIPS process. CLIPS permits the agent modules to import and export object values from the CLIPS 'Main' module that holds the data-blackboard. A major advantage of this implementation is that on each computer the same physical COOL objects and CLIPS facts are used by every module. For each world, ***private***, ***shared***, and ***display***, only one copy of a semantic network entity exits in the CLIPS process and that copy of the entity is addressed by every agent that wishes to use it.

The CLIPS process provides facilities to manipulate a 'stack' of the modules that are to receive CPU cycles. A 'focus' call is used to select the next module to execute rules from its agenda, allowing a specific module or the module at the top of the stack to be chosen. Since 'focus' can be called from the right hand side of rules as well as within user functions, there is literally no limit to the control that can be exercised to select the module that should execute. However, a simple round-robin algorithm works exceedingly well in the prototype system. The private and shared world agents execute concurrently. While it would be possible to prohibit the agents from executing in one or the other worlds, there did not appear to be any advantage from the user's point of view in implementing this kind of control. It was considered more important that the user be free to call up any reports and generally interact with any of the three worlds at any time.

7.2.3 The Agents

The FEAT prototype implementation employs agents written in the CLIPS expert system shell language. Individual agents on a specific CPU are encoded within one CLIPS process as distinct CLIPS modules. Modules provide independence of agent rule-sets, while permitting the agents to share references to the same facts, templates, and objects. Agent activity is displayed through icons, and visual cues are used to identify when an agent is working, as well as when a warning or communication from the agent is available. The current FEAT agents include the following domains:

Cost: A general cost agent.

Lift: An agent specific to air transportation reasoning.

Time: An agent that primarily monitors relations between activities that involve the time at which they are planned to take place.

Location: An agent that handles information specific to a location, for an activity.

Availability: An agent that considers combinations of criteria associated with availability.

Readiness: An agent that evaluates and checks the 'readiness' of units.

While it is recognized that in a final end-user system there would need to be more agents, the six agents included in the FEAT prototype system are sufficient to demonstrate the nature and potential capabilities of a multi-agent system in support of military planning and execution applications.

7.2.4 Communication Facilities

There are two areas of communication in FEAT. The first area is the communication that takes place between the CLIPS and C++ processes. On a specific FEAT workstation (i.e., CPU) this communication ensures that the semantic network copies in the CLIPS and C++ processes are in concert. The second area is the communication between and among the CPUs. A set of communication procedures makes it possible to transmit and receive COOL object information in equivalent C and C++ representations. This communication is external to the CLIPS process and is a modification of the CMS message passing system developed by the CADRC for previous implementations of ICDM (Pohl 1995, Myers and Pohl 1994). This new version of CMS provides functions that can be called in C, C++, or CLIPS to communicate object information to any agent or any group of agents in the system. It also provides for the reception of messages and assimilation of messages into the native environment of the receiving component. Essentially, it establishes a common protocol for the transmission of formatted messages.

In addition, the communication component includes the code that handles the mechanisms through which remote computers are involved in an active FEAT session. This includes facilities to provide the dynamic connection to a session, the entering and leaving of a remote communication group, transmission of remote data, transmission of noteboard data, monitoring of active CPUs, and proper exiting of processes. In particular, the remote object communication and noteboard facilities discussed later (see Sections 7.2.6 and 7.2.7) are implemented through CMS communication calls.

The object communication facility permits an object defined within one FEAT system world to be communicated to selected worlds of remote FEAT systems. For example, an activity object can be sent from the private environment of one system to the shared environment of another. However, there is no particular limit to the number of objects or characteristics relating to a single object that can be communicated. In other words, a single attribute of an object or the attributes of several objects the entire semantic network can be transmitted.

7.2.5 The User-Interface and Report Facilities

The user-interface is implemented as a set of Motif widgets (Gregory 1992). It provides dynamic windows that permit the user to 'point and click' to select the primary available options (Figure 64). By 'wrapping' the widgets in C++ class definitions a library of commonly used widgets is produced for the implementation of the user-interface.

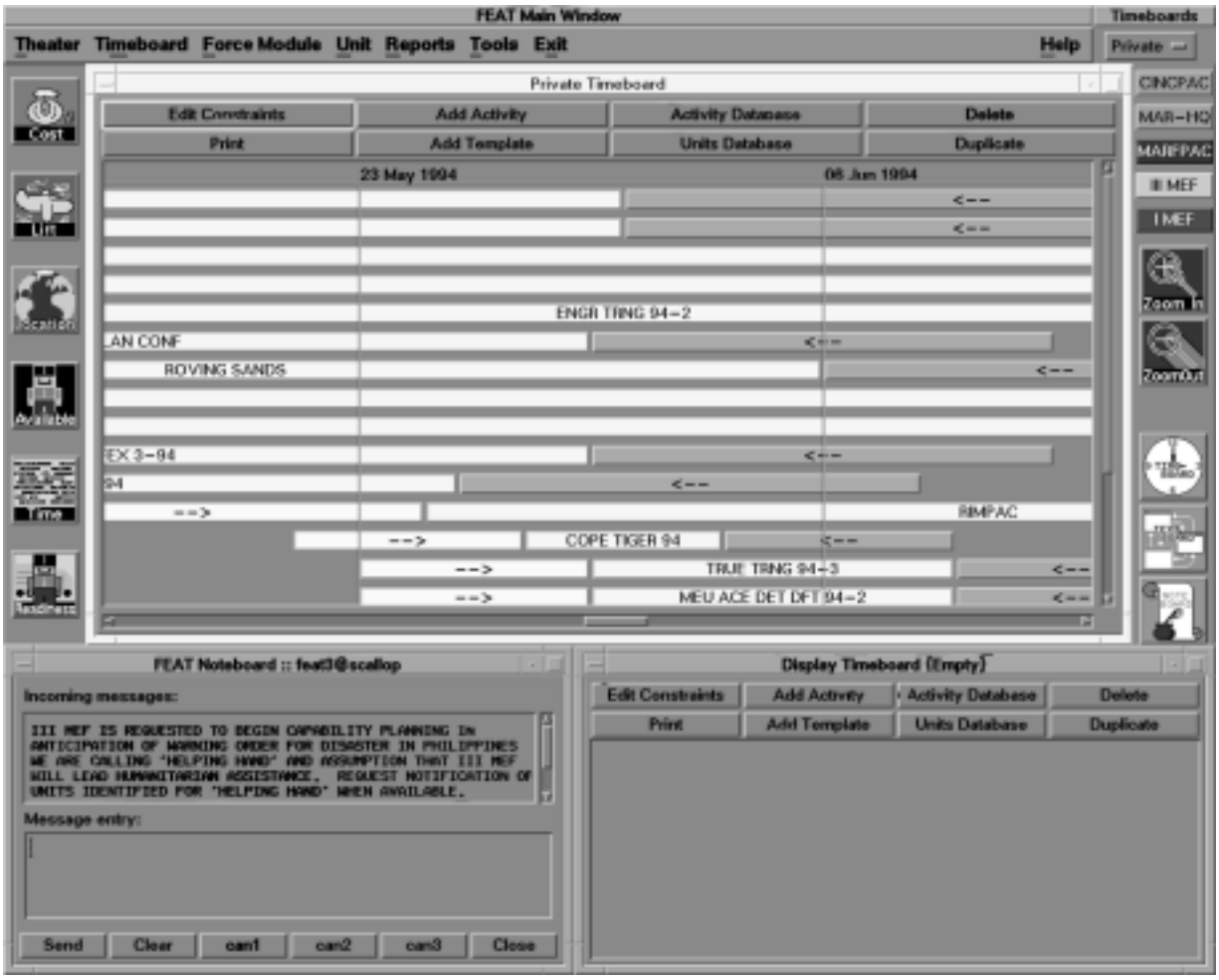


Figure 64: Main Screen of the FEAT User Interface

Several table, graph, and chart display facilities are used to provide summaries of the current problem state to the users. Graphic reports are provided by XRT widgets (KL Group 1995) and include facilities for the generation of bar graphs, line graphs, tables, and surfaces.

7.2.6 The Remote Object Communication Facility

To facilitate a better understanding of the operation of the FEAT system, the actions associated with a related set of remote communications of objects can be described as a 'textboard session'. A textboard session is initiated by any FEAT user when he or she selects other FEAT users as textboard group members. The information that is sent in a textboard session is referred to as the current textboard, or simply 'the textboard.'

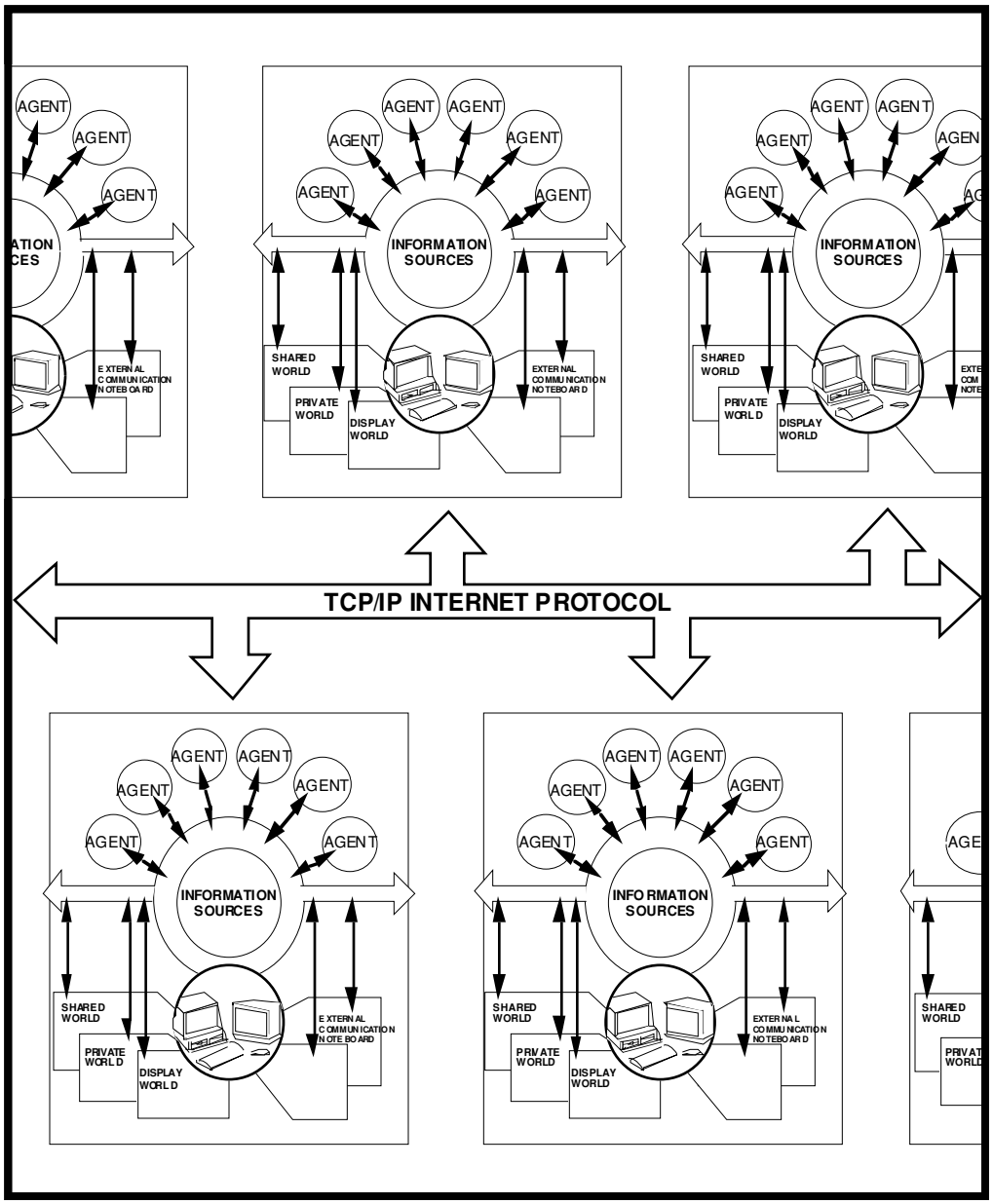


Figure 65: 'Textboard' Transmissions Among Multiple FEAT Workstations

The most common textboard session occurs when one FEAT user wishes to share the plans for a new activity with another user. In this case the sender transmits the activity object and its associated unit objects, describing the units that are involved in the activity

to the display environment of the receiving user. If the receiving user wishes to see how the agents in the receiving system would react to the new activity, the primary values from the activity and associated units are sent to the receiving unit's shared environment as well. The primary values are those that agents use to generate inferred or dependent values. When the primary values are received in the shared world the agents of the receiving environment generate whatever may be inferred as dependent values. This permits the receiving unit (i.e., FEAT workstation) to determine how its agents would react to the proposed activity (Figure 65). The transmission of values to the display world of the receiving unit permits the user to view the results obtained on the remote system and compare those results with what is obtained in the receiver's shared world.

If all associated data and agents are identical on the sending and receiving systems, the dependent values that result from the assertion of primary values will be identical. But differences in local data and/or local agents might result in a different dependent value being produced from that generated on the sending unit. While this difference might be due to an error, it may also be the result of important proper differences among the data or agents on the two units. It is important to note that this capability eliminates the need to guarantee absolute agreement among the computers and in fact permits variations in data and agents to exist on different units.

7.2.7 The Noteboard Facility

A noteboard, or message board, permits users at different stations in the network to transmit text to groups and/or selected stations. In particular, the noteboard is used to authorize the communication of textboard information and to transmit comments with regard to the information received. Specifically it might be used to reach agreement among the human collaborators as to what values in a collaboration should be changed.

Icons at the right side of the main FEAT screen identify the active FEAT workstations with which communication can be established (Figure 64). These icons are color coded. Text displayed on the noteboard and most of the information received in a textboard session is displayed in the same color as the icon of the sending unit. This makes it easy for users to differentiate between noteboard message sent from a variety of FEAT workstations.

7.3 Continued Work on FEAT4

The CADRC is currently extending the FEAT system to support the Sea Dragon program of the US Marine Corps (Krulak 1996), under the direction of the Commandant's Warfighting Laboratory (CWL). This version, referred to as FEAT4, focuses on decision-support for mission analysis, planning, and execution activities that occur in the various sections of the Extended Combat Operations Center (ECOC).

Multiple FEAT4 workstations, distributed within the ECOC, are linked to the base Marine Corps C4I (Command, Control, Communications, Computers, and Intelligence)

system through a JMCIS server. Information entering the JMCIS Track Database Manager (TDBM) from various sources (e.g., small hand-held Newton computers operated by Marines in the battlefield) is transmitted in the form of objects to each FEAT4 workstation. There, expert agents interacting with each other and the user monitor changes in enemy and friendly forces positions, and assist in the development and analysis of strategies and plans.

The FEAT4 system also incorporates an Object Command Language (OCL) that allows users to conveniently enter OPLAN (Operations Plan) components into an object-based knowledge base. By utilizing a simple 'point and click' user-interface operators are able to translate textual OPLAN statements into object constructs. For example, the OPLAN sentence ***"During Stage 2 the SPMAGTF will resist enemy aggression by attacking and neutralizing enemy forces."*** would be entered into FEAT4 by the user as a **Commander's Intent** object consisting of the following components:

Reference:	ANNEX	B
	SECTION	Situation
	SUB-SECTION	General
Element:	SOURCE	CJTF
	EXECUTOR	SPMAGTF
	TARGET	enemy
	WHEN	stage 2
	ACTION	attack
	ACTION	neutralize
	OBJECTIVE	resist aggression
Qualifier:	DURING	

This OPLAN knowledge base, together with several databases (e.g., weapons and equipment, munitions, target types, friendly assets, etc.), serve as context for the agents and the users as they analyze, plan and evaluate alternative courses of action.

Central to the FEAT4 architecture (Figure 66) are two components, the semantic network and the agent kernel. The semantic network contains the current state of the problem in object format (i.e., high level representation). Changes to the current state that are received on a continuous basis from various sources (e.g., TDBM, users utilizing the OCL) drive the agents as they respond to requests for services. The agent kernel coordinates the discourse among the agents, and supports both direct (i.e., agent to agent) and broadcast (i.e., one agent to all other agents) communications.

Apart from the JMCIS interface, the FEAT4 workstations are also connected to a textboard-type Shared Net that adds a desirable level of redundancy to the communication environment. Unfortunately, in the first version of the Sea Dragon C4I implementation the Shared Net will lack internal object representation capabilities that are adequate for agent inferencing purposes and will therefore serve mostly as a back-up message passing facility.

In subsequent versions it is proposed to add natural language processing capabilities that would elevate the Shared Net to an information source, as compared to the data source currently provided by the JMCIS TDBM.

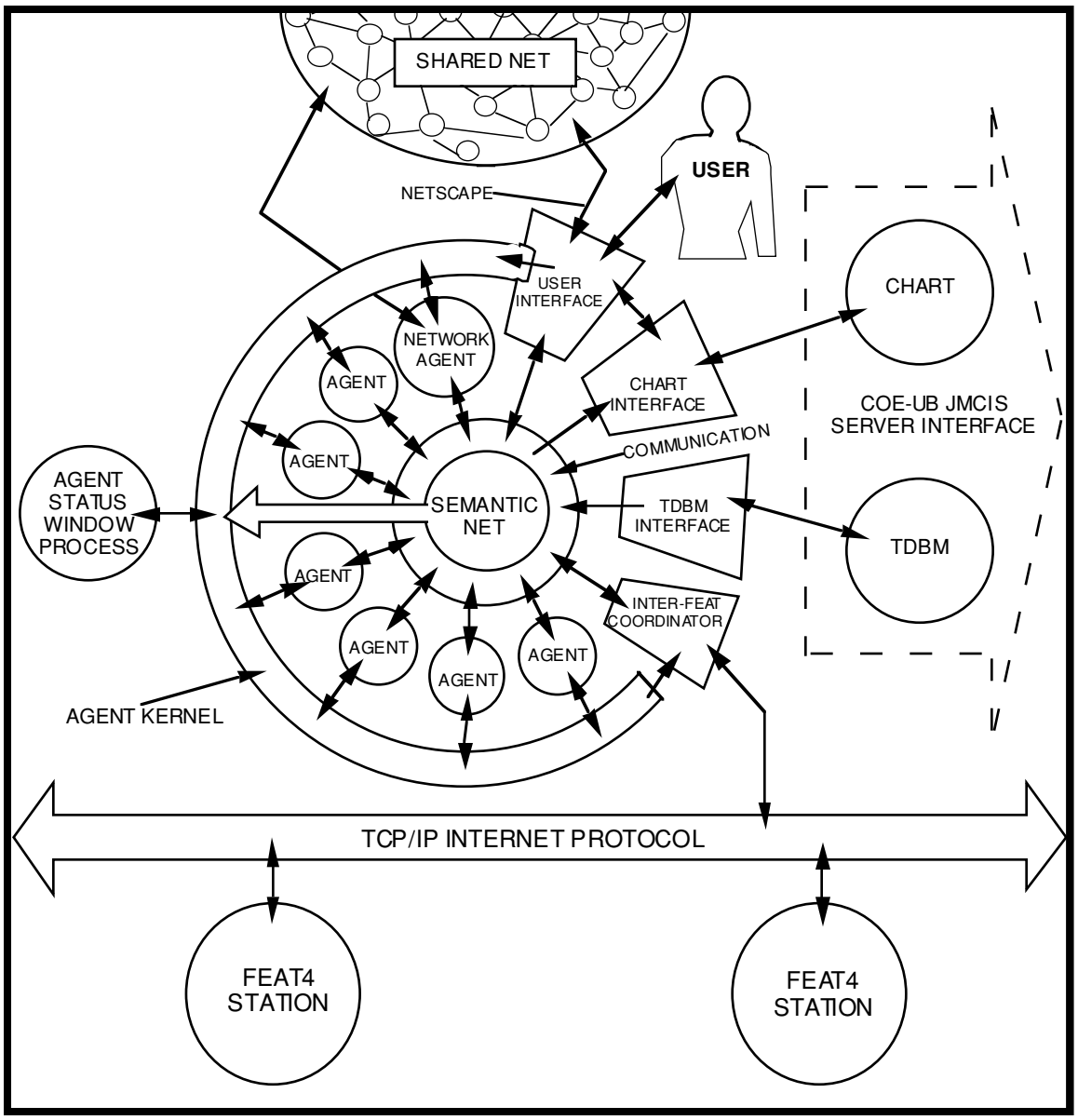


Figure 66: Schematic Representation of the FEAT4 Architecture

8. CIAT: Facilities Management with Multiple Service-Agent Systems

In 1996, the Collaborative Agent Design Research Center (CADRC) working jointly with CDM Technologies, Inc., undertook the development of a proof-of-concept implementation of the ICDM model in the facilities management area. The objective of the CIAT (Collaborative Infrastructure Assessment Tool) is to provide a decision-support system that will allow NAVFAC, Fleets, CINCs (recently renamed Combatant Commands) and Base Staff Civil Engineers to collaboratively recommend and assess the impact and effectiveness, from a mission perspective, of investments of the US Navy's multi-billion dollar facilities/installation budget.

8.1 Port Resources Scheduling as a Complex Problem

CIAT is an ICDM-based decision-support system that identifies conflicts and assists human decision makers in complex problem situations that arise in infrastructure assessment and utilization situations. The facilities management application shares many characteristics that are common to complex problem situations, such as: information overload; complex interrelationships; uncertainty; severe constraints; and multiple decision-makers needing to collaboratively reach consensus (Figure 67).

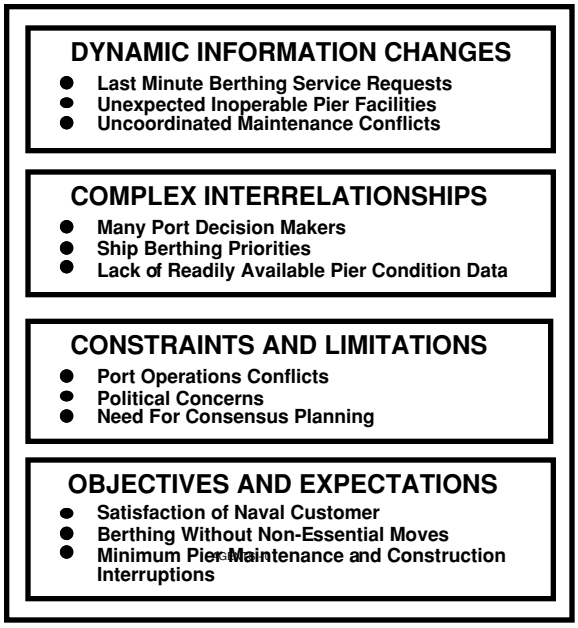


Figure 67: The Complexity of Ship Berthing

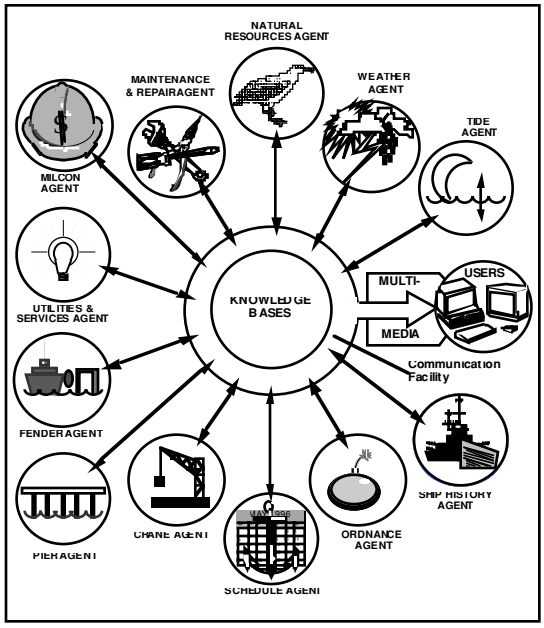


Figure 68: CIAT Decision Domains

US Naval Stations typically have a limited number of piers available to homeport US Navy ships and regularly service foreign navies. Military pier and ship schedulers have a difficult task. There are frequent last minute berthing service requests and at least an equal number of unexpected equipment and facility failures. Once a commitment has been made and a large ship (e.g., aircraft carrier or destroyer) is berthed at a particular pier, it is a relatively expensive and time-consuming proposition to move that vessel to another pier.

Schedulers are continuously confronted with uncoordinated maintenance conflicts. Ports are relatively large installations involving many parties that play different roles and have unaligned agendas. In this environment it is not uncommon for one group to schedule activities that are in direct conflict with the proposed activities of one or more other groups.

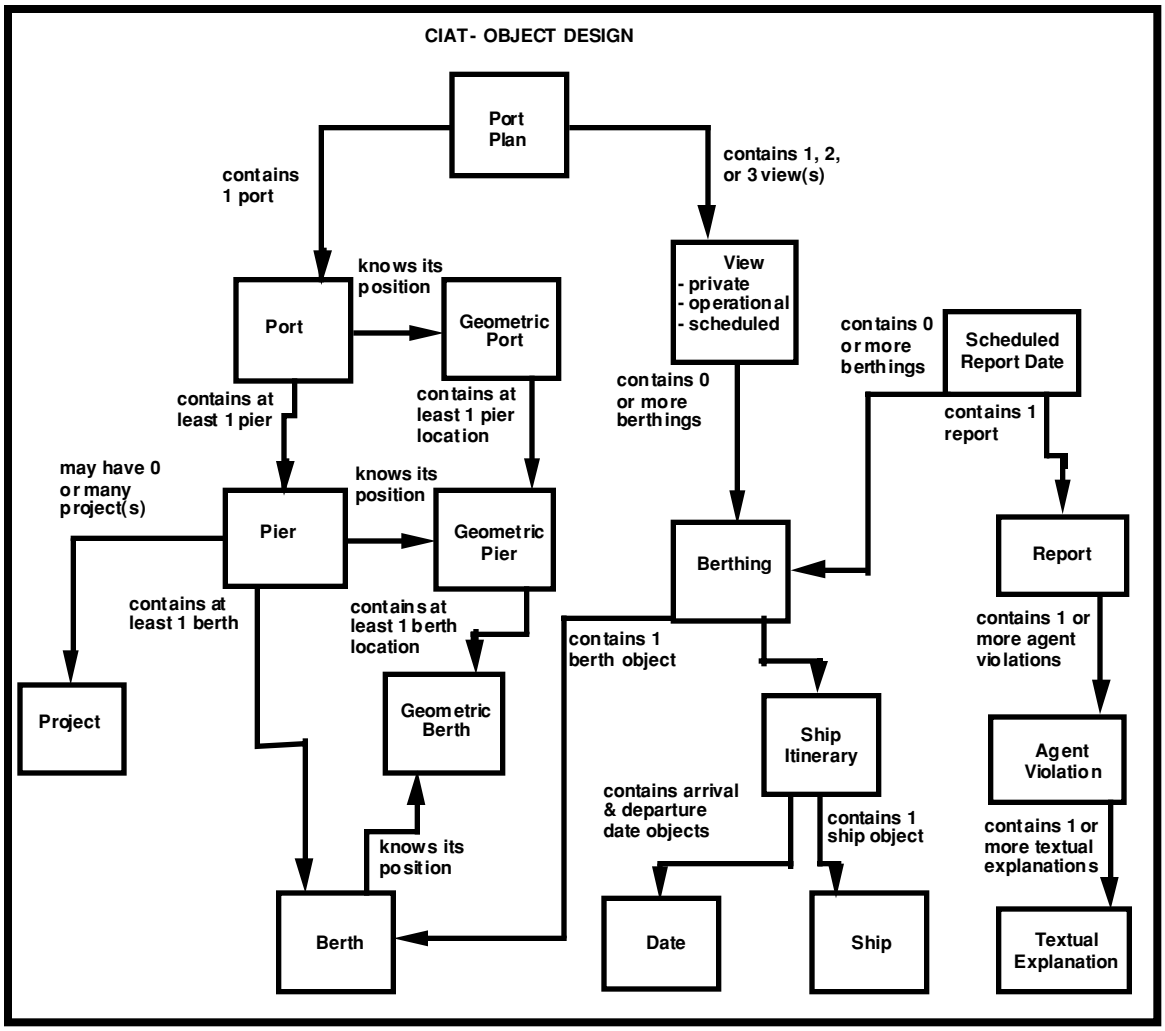


Figure 69: Object Relationships of the CIAT Application

Currently with the availability of only limited, largely stand-alone computer facilities (see Sections 2.3.1 and 2.3.2, and Figure 5), there is little collaboration among these groups leading to frequent service conflicts and wasted resources. There is a need for an integrated decision-support system that is designed to both provide assistance in the analysis, evaluation and planning tasks that occur within a particular group and facilitate the collaboration among several groups. In fact, the value of such a system is likely to be most pronounced in its encouragement of consensus planning involving all of the major players.

8.2 System Description and Architecture

The architecture of CIAT is very similar to the FEAT system described in Section 7. It therefore suffices in this section to address operational considerations, in particular the functions of the various agents (Figure 68). CIAT is an object-based system that employs expert system technology for analysis and evaluation. The representation of knowledge is in objects and object relations that describe the real word entities (e.g., piers, cranes, utilities, ordnance, etc.). Pier schedulers reason about these entities and how they relate to one another. Figure 69 shows a relationship diagram of the object entities that are central to the pier scheduling activity. The analysis of actual scheduling situations is performed by a number of agents that look at the current state of the problem situation from their individual perspectives and flag any violation of scheduling rules and guidelines. In the proof-of-concept system emphasis has been placed on conflict identification. Future work will extend these agent activities into collaborative conflict resolution, even though it is our current philosophy in the CADRC to maximize user involvement in the final resolution of conflicts.

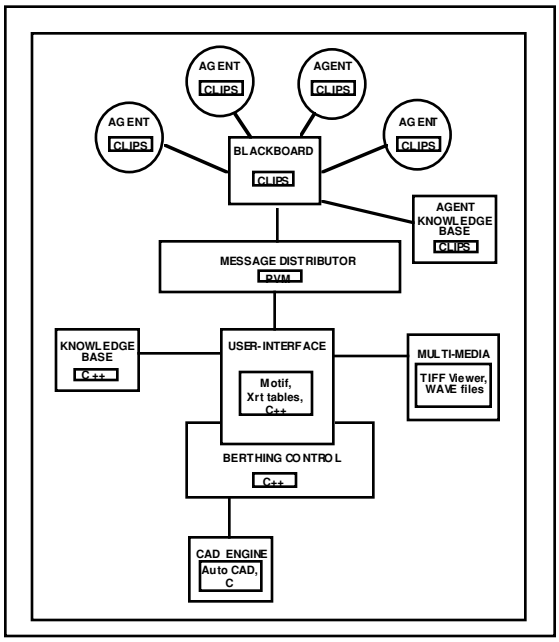


Figure 70: CIAT System Architecture

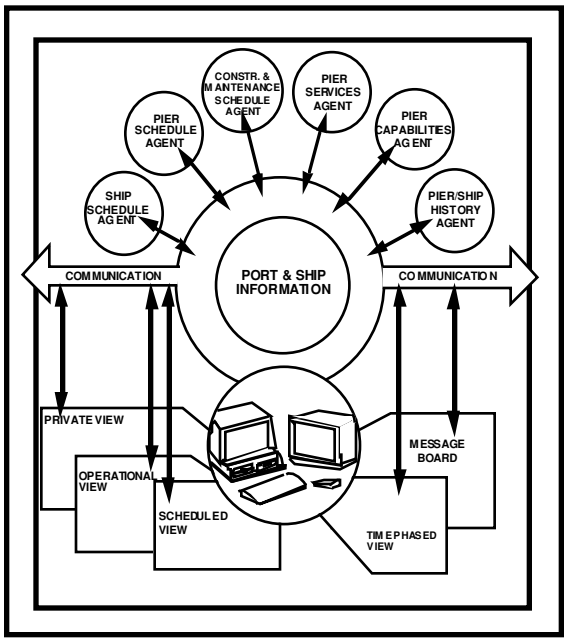


Figure 71: CIAT Application Components

The system comprises four main parts: graphics; user-interface; agents; and, internal communication facilities (Figure 70). The graphics component supports the processing and manipulation of ship and pier drawings, as well as functions for modifying the currently displayed drawing to correspond to problem states on past and future dates. The drawing engine that is used in the current version of CIAT is AutoCAD (AutoDesk 1993), although the user does not have direct access to the AutoCAD drawing environment. Instead, the user-interface seamlessly sends drawing instructions to the AutoCAD process whenever the user initiates a drawing action. This is necessary because AutoCAD does not have the facilities to manipulate the high level object representation that is required within the CIAT application.

The user-interface component provides all of the functions that the user utilizes to manipulate the system. It is the part of the system that drives the logic of the agents based on the actions of the user at any point in time. The layout of the user-interface is shown in Figure 72. The agents represent the logic base of CIAT. They incorporate knowledge about the scheduling problem and analyze the current schedule proposal according to this knowledge. The knowledge is represented in the form of rules that are triggered by the existence or the non-existence of certain objects. All of the agents are coded in the CLIPS expert system shell language (NASA 1992) and are modules of a single CLIPS process. This process also includes an internal blackboard that coordinates the activities of the agents and routes messages to the agents (Figure 70).

The communication system functions at two levels. At the first level, all of the internal components of CIAT communicate using the CMS facility of the ICDM model (Myers and Pohl 1994, Pohl 1995). Internal communication is required for exchanging information about the status of the system for user-interface operations. CMS activity is transparent to the user. In other words, all communication functions (i.e., create, update or remove objects) are triggered automatically according to the appropriate situation. In addition, at the first level, internal communication and coordination facilities provided by the ICDM kernel allow agents to collaborate with each other and the user in a blackboard-like environment. However, all agent actions are driven opportunistically by changes in the problem state without any form of scheduling or time-stamping. As discussed previously in Section 4.1, the ICDM communication facilities incorporate preemption (i.e., interrupt) mechanisms that are built on top of the Parallel Virtual Machine (PVM) inter-process communications library (Beguelin et al.1991).

At the second level, multiple CIAT workstations communicate through TCP/IP protocol networks allowing the transmission of solution proposals and plans, as well as text messages from one CIAT workstation to another specific workstation or all other workstations (Figure 71). On each CIAT workstation the user is able to plan in ***private*** (i.e., local), ***operational*** (i.e., shared) and ***scheduled*** (i.e., accepted) worlds representing those levels of planning. Agents representing critical areas of expertise, such as berthing, facilities scheduling, infrastructure capabilities and maintenance, and construction issues, continuously assist the users in identifying and resolving conflicts. The agents are able to differentiate between plans that are concurrently under preparation in the three different planning worlds (i.e., private, operational, and scheduled) and assist uniquely within the boundaries of each world.

8.2.1 The Agents

The agents consist of database knowledge and the rules for manipulating this knowledge in respect to specific domains. The database contains primarily static information, such as ship characteristics, pier facilities and capabilities, ship/pier history, and so on, which for the purpose of the CIAT proof-of-concept system is stored in flat files. Future more

extensive implementations of the CIAT application will likely require a more sophisticated database management system. The agents that operate on database information include:

Ship Schedule Agent:

Representation: Ship objects; including ship description (class), geometric (dimensions, draft), non-geometric (power needed, etc.) attributes, ship berthing; including ship itinerary (arrival time, departure time, and duration of stay) and a proposed berth.

Functionality: A ship is assigned a berth for docking and services required during its stay. The berthing object includes the dates of arrival and departure (duration of stay). The agent registers its information and communicates it to the Pier Scheduling agent. In an assisted-scheduling mode this agent will suggest a berthing based on the availability of berths, triggering the other agents in the same way as the user-scheduling mode.

Pier Schedule Agent

Representation: Pier objects; pier name and services offered.

Functionality: A ship is assigned a berth on a pier. The berth is checked against the current pier schedule. If other ships occupy the same berth during any day of the requested berth, a violation is flagged. In assisted-scheduling mode the agent lists available piers (or berths), after consulting the Ship/Pier History agent to determine whether the selection of the first choice of berth could be influenced by the berth assigned to this ship for its previous visit.

Construction and Maintenance Schedule Agent

Representation: Pier objects; pier name. Project objects; project name, duration and location.

Functionality: A pier assignment is received. The agent interrogates the construction/maintenance schedule for pier activity. If the assigned pier is scheduled for work during the specified period of time the agent indicates a violation.

Pier Services Agent

Representation: Pier objects; pier name and services offered.

Functionality: A request for services is received and matched against the services available at the assigned pier (i.e., as determined by the pier schedule agent). If any of the services are not available (i.e., non-existent or unavailable due to maintenance or otherwise) the agent indicates a violation.

Pier Capabilities Agent

Representation: Pier objects; pier name and characteristics. Ship objects; name and characteristics.

Functionality: When a berthing object is received, the agent matches the services needed with those available on the specified pier and flags a violation if the pier does not offer one or more of the required services. In an assisted-scheduling mode this agent looks at the piers and lists those that can offer the required services.

Pier/Ship History Agent

Representation: Pier objects; pier name and services offered. Ship objects; name and characteristics.

Functionality: When a ship is assigned a berth in either user-scheduling or assisted-scheduling mode this agent checks the history file and reports its findings (i.e., has the ship has ever been assigned there; was it assigned there for a number of days; or, has it never been assigned to that berth but to another berth on the same pier). In the assisted-scheduling mode, when a ship request for berthing is received, the pier that serviced it last is proposed first to again provide the required services.

The following additional agents are proposed for future extensions of the CIAT system:

Financial Analysis Agent

This agent will analyze the cost of assigning a ship to a specific berth (i.e., crane services, extra power cables, transportation of crew members, utility costs, etc.).

Quality Control Agent

This agent will check the suggested berthings against rules and regulations of operation and flag any inconsistency. It will also check the validity of data, such as berthing requests, by time-stamping all information coming into the system and invalidating dated requests after a specified period of time.

Environmental Control Agent

This agent will examine the impact of ship berthings on environmental concerns and regulations and present problems and potential issues to the user. It is proposed that this agent would also keep a history file of environmental violations and protests and attempt to match future facility activity patterns to previous negative situations.

8.2.2 The User Interface

The major components of the user-interface are shown in Figure 72. In the center of the main CIAT screen is a scaled drawing of the pier configuration (i.e., NAVSTA San Diego,

CA) which serves as the demonstration scenario for the proof-of-concept system. As the hour and the day change, the user can view the corresponding changes in ship placement at individual berths. The individual ships are represented as scaled, plan-view outlines appropriate to each class with ship names within each outline.

On the left side of the screen are status icons for the various agents. Whenever an agent is active a yellow border will appear around the corresponding icon. In the current system icons are included for the following agents: ***Ship Schedule agent*** - to monitor the needs of ship arrival and departures; ***Pier Schedule agent*** - to monitor the availability of berths; ***Construction and Maintenance Schedule agent*** - to coordinate the construction and maintenance activities with the ship and berth schedules; ***Pier Services agent*** - to coordinate the availability of services with ship and berth schedules; ***Pier Capabilities agent*** - to coordinate the pier capabilities with ship needs; and, ***Pier and Ship History agent*** - to propose ship placement based on historical activities.

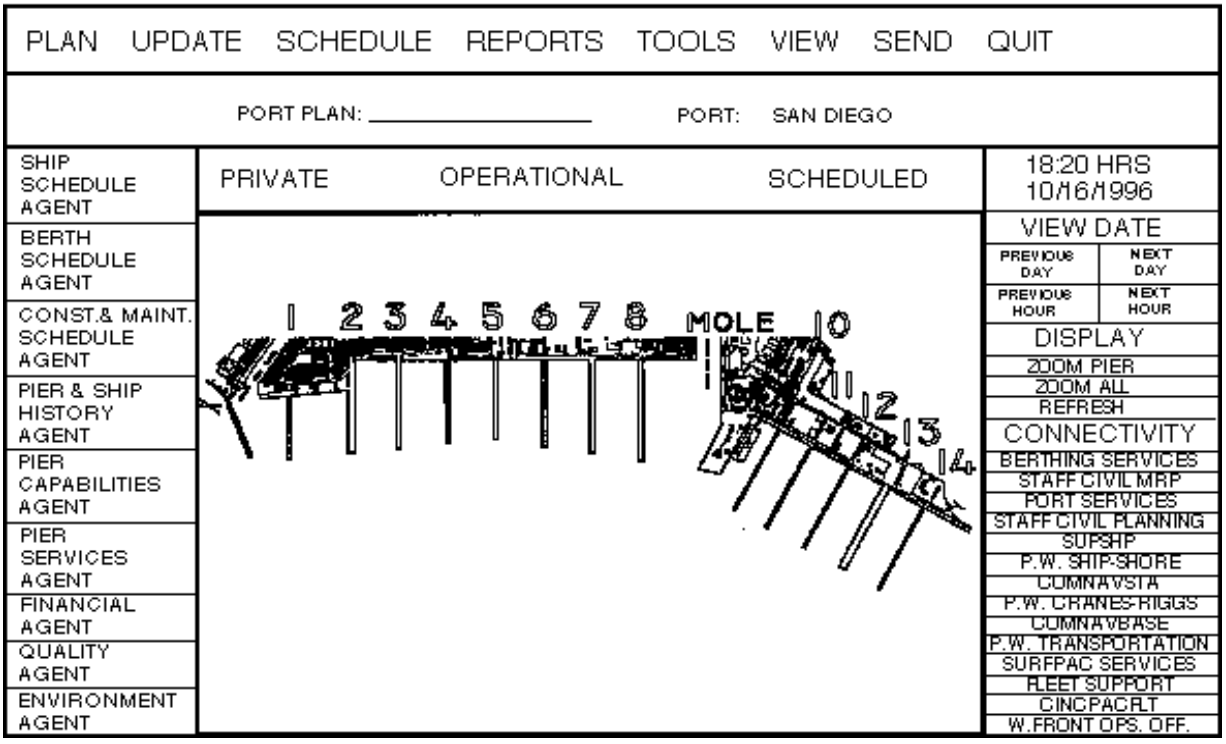


Figure 72: CIAT User Interface Layout

At the upper right side of the screen are a number of utility functions including: the currently displayed day and time; the options to change the day and time; and, display options to zoom to a single pier, zoom to the entire set of piers or refresh the screen. On the lower right side of the screen is a list of typical port decision-makers (i.e., groups) that might be simultaneously connected in a collaborative problem solving session. When a particular member or group is connected the boxes including that name are highlighted.

At the top of the screen from left to right are the following principal CIAT options: ***Plan*** - to initiate a new, or use an existing planning session; ***Update*** - to update data files for pier construction and maintenance, pier and port services, and ship arrival and departure times; ***Schedule*** - to assign and un-assign a ship to a particular berth, including both user-scheduling and assisted-scheduling modes; ***Reports*** - to generate a number of reports including the schedule of ships, schedule of berths and piers, schedule of pier maintenance and repair, and schedule of intermediate maintenance availability servicing; ***Tools*** - for textual notes on the pier/ship drawing, tidal calculations, display of detailed pier drawings, and display of detailed ships drawings; ***View*** - to work with multiple views of the ships/berthing schedule; ***Send*** - option to send textual messages and the operational view to one or more decision-makers; and, ***Quit*** – to terminate use of the system.

All CIAT users have an accepted ***scheduled*** view, which represents the view that has official approval. It is expected that individual users or groups will investigate options for future alternatives using the ***operational*** view. This will involve all of the data from the scheduled view with proposed changes. Individual users also have the option of investigating berth assignments using a ***private*** view which may or may not become an operational view. Utilizing the view concept, a user or a group of users have the ability to investigate multiple future options without changing the accepted or scheduled view. At some point a decision-maker in authority must exercise his or her authority to move an operational view into an approved scheduled view. On the screen, immediately below these CIAT options is a banner identifying the current port plan, the current port of concern, and the current view (i.e., private, operational or scheduled).

9. KOALA: Architectural Space Planning with Object-Agents

Design is indeed an ubiquitous activity. In the physical world every artifact, whether it be a coffee maker, a miniature silicon sensor for invasive blood pressure monitoring, a ship, an automobile, or a building, is the result of some kind of design activity. However, design is concerned not only with the creation of artifacts. Any problem solving situation in which there exists an element of the unknown, such as lack of information or incomplete knowledge of the relationships among variables, involves an intellectual effort that can be categorized as design.

It follows that design is more than the rote calculation of an algorithm or the copying of a known process, although either or both of these may be useful design tools. For an intellectual endeavor to qualify as a design activity it must involve relationships that cannot be totally defined and appear to pose some degree of conflict. Indeed, the resolution of actual, perceived and potential conflicts is a fundamental ingredient of all design endeavors.

9.1 The Complex Nature of Design

Typically, design requires decisions to be made among several imperfect alternatives. It is in the nature of those decisions that designers will often find the need to supplement logical reasoning with intuitive feelings about the problem situation that can lead to creative solutions and new knowledge. As a rule such new knowledge cannot be logically deduced from the existing available knowledge and is validated only after the solution has been discovered and tested. In this respect design is not unlike the decision-making activities that occur in a wide range of complex problem situations that have to be dealt with in many professional fields such as military planning, management, economics, law, medicine, and transportation.

The quality of design solutions will vary significantly as a function of the human and information resources that can be brought to bear on the solution process. Designers often make errors in judgment during the earliest de sign stages that require costly and time-consuming corrections down stream (Figure73). Unfortunately, these later corrections can lead to other compounding errors if the original intent of the designer is not known.

The principal causes of design errors are lack of experience and knowledge of the designer, and inadequate integration and coordination of the various parties that are involved in the design to product cycle. It is therefore not surprising that a great deal of interest and research activity has been focused in recent years on supporting the design activity in a computer-assisted environment. However, the effective realization of this objective has proven to be a much more difficult and elusive undertaking than first anticipated. The reasons are related not only to the ill-defined nature of the activity (Rittel and Webber 1984, Simon 1984), but also to the inadequacies of the representational and operational models that have been used as the framework for computerization. At the core of these inadequacies have been the issues related to the representation of knowledge within the computer and the interface between the human designer and the computer-based

assistance components. These issues have been discussed throughout this Technical Report (see in particular Sections 1.4.5 and 3.2.2 for representation issues, and Sections 1.4.1, 1.4.9 and 3.2.6 for computer-user interface issues) and therefore will not be reiterated in this Section.

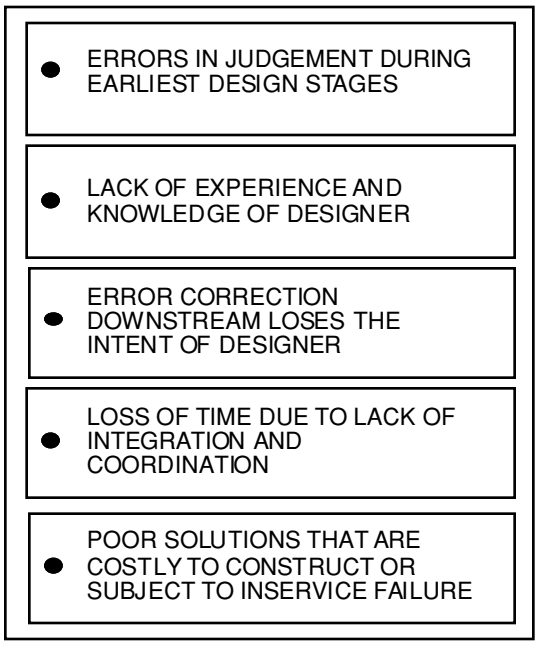


Figure 73: Typical Failures in Design

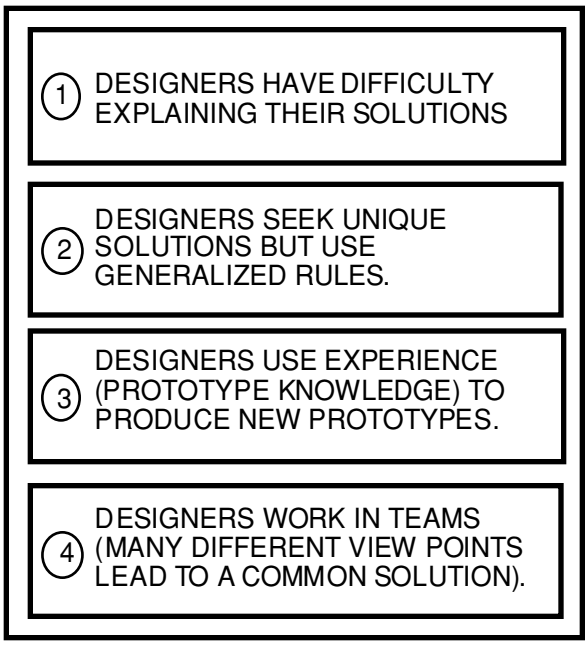


Figure 74: Schon's Design Tensions

Although reasoning is an essential functional component of the design activity, it would be misleading to suggest that design is performed in an entirely methodical manner as a set of sequential transformations. Schon (1988) has drawn attention to four fundamental tensions in design (Figure 74). First, designers typically have difficulty articulating the knowledge and methods that they apply to the design activity. While the design solutions that they produce are often ingenious, the subsequent explanations of the designer are less convincing. If the knowledge the designer holds cannot be made explicit then what kind of knowledge is it, how is it retained, and how can it be accessed when needed?

Second, there is the apparent paradox between the designer's quest for a unique solution and the requirement of general rules to reason out that solution. Third, designers accumulate knowledge from one project to the next. If they apply this prototype knowledge derived from past projects to produce design solutions, how can they ever generate new prototypes? Fourth, architecture and engineering design is a team effort. The team members have pluralistic backgrounds, interests, and agendas. Yet, they normally agree on a common design solution.

While designers will employ sequential reasoning in short bursts and over somewhat longer periods to explore and evaluate solution alternatives, the generation of the alternatives themselves is often neither sequential nor logical. The latter is characterized by the spontaneous introduction of apparently unrelated thoughts, associations that

transcend purely logical relationships, whimsy, and tacit understandings that defy explanation. In other words, intuition appears to play a major role in most design endeavors.

9.2 Agent Types and Interactions in KOALA

The KOALA (Knowledge-Based Object-Agent Collaboration) system extends the service-agent implementations discussed in Section 6 (ICODES), Section 7 (FEAT) and Section 8 (CIAT), by introducing two additional agent types: object-agents that represent high level data objects such as the spaces in a building; and, monitor-agents that act as facilitators during interactions.

Service-agents have expert knowledge in narrowly defined fields and typically provide advisory services based on their knowledge. In military mission planning, as exemplified by the FEAT prototype system (see Section 7), such knowledge domains include: cost; time; transportation; readiness; location; and, availability. For example, in the FEAT decision-support environment the ***Lift*** agent services requests for determining the availability and suitability of aircraft for transporting troops to the theater.

In KOALA, ***Space*** agents are a specific instantiation of a more general type of agent that can represent the interests of a high level object that plays a significant role in the decision-making process of the application environment. In the building design application, spaces or rooms play an important role in the development of floor plans. The architect manipulates spaces as complex data objects with strong relationships to each other and equally important relationships to data entities that are related but substantially different in nature (e.g., occupant activities, privacy, security, etc.). The ability of the human designer to reason about the relationships among complex data objects is an essential part of the decision-making process that underlies the design activity.

In multi-agent systems such as FEAT a semantic modeling approach (Myers et al.1993) is employed to define a common vocabulary that serves as an internal high level representation of real world objects, such as Marine Corps units, theater locations, and aircraft. This approach provides a workable basis for service-agents to monitor the evolving solution plan and communicate with each other and the human decision-makers through an internal collaboration facility. However, the relationships among data entities are represented only to the extent that the service-agents view the solution state from their respective knowledge domains with special conflict identification agents attempting to reconcile these, often conflicting, views. Therefore, the success of this approach must rely heavily on predefined knowledge that is embedded in the agents, and user interaction (i.e., the intervention of the users to maintain and prioritize relationships as a reflection of their intent).

A different approach is to treat the objects that play a major role in the problem environment (e.g., building design), not as passive data entities, but as active agents. Such object-agents can utilize communication capabilities to dynamically create and maintain

relationships to other object-agents. Potentially, this would appear to be a significantly more promising approach. Such an environment allows a complex problem system to be decomposed into sub-problems without diluting or losing relationships. To the contrary, relationships are greatly strengthened through the dynamic nature of communication in a collaborative environment. ***Space*** agents then are object-agents that have knowledge of their own nature (i.e., essentially the same descriptions that are contained in a *space* data-object) and the ability to interact with other agents through their communication capabilities. Utilizing this knowledge as a basis for developing their interests and desires, ***Space*** agents attempt to satisfy such concerns by acting on their knowledge, gaining additional information, and requesting services from other agents. In this highly collaborative environment there is a need for facilitators to detect conflicts and moderate arguments among object-agents. This role is assumed by ***Monitor*** agents. In summary, KOALA supports an agent taxonomy that includes service-agents and object-agents, as follows:

Designer **Agent:** KOALA recognizes the human designer (i.e., the user) as the most intelligent agent in the computer-based design environment. Capable of a wide variety of cognitive skills ranging from in-depth analysis to highly abstract conceptualization, the human designer essentially orchestrates the evolving design solution. Unique to this agent is the notion of intent. KOALA represents such notions with the provision of a Designer agent. It is the responsibility of this agent to not only acquire the designer's intent, but to also maintain its reflection in the decisions being made by the agents in the system. Intent may be explicitly expressed in the form of design criteria, such as performance requirements, or implicitly hidden in decisions that are influenced by vaguely defined perceptions and subtle nuances. In the design activity, the notion of intent is essentially embedded in the strategy employed by the designer.

Service-Agents: KOALA includes several service-agents that represent expertise within specific knowledge domains. Each agent provides expert evaluations and consultation based on its particular area of aptitude. This analysis is largely driven by prototypical knowledge. Based on such knowledge, an educated comparison can be made between the various attributes and characteristics comprising the current solution and those commonly associated with design elements of a similar nature in a related environment. The exact set and depth of domains represented depends on the context in which the application is to be employed. In KOALA, service-agents offer other agents, including the user, a domain specific pool of expertise capable of providing supporting design solutions given a certain set of conditions. Once invoked, these domain agents may employ the services of other agents (including the user) to perform the requested analysis.

Space **Agent:** A building space can be defined as a physical volume bounded by one or more physical surfaces or implied boundaries. Further, a space is governed by a set of constraints and guidelines, which can be made available through prototypical information. In KOALA such prototype knowledge forms the basis for agent evaluation, and is used by a ***Space*** agent to establish a set of interests and desires relating to a particular space.

With the addition (by the user) of any space into the evolving design solution a ***Space*** agent is created and associated with that particular space. The sole purpose of a ***Space*** agent is to represent the interests of its space counterpart (i.e., acting as a mentor for that space). Consequentially, each ***Space*** agent views the world (i.e., the solution space) from its own, potentially biased perspective. However, such biases are an important ingredient of a truly autonomous environment. As in human group collaborations these biases reflect the variety of viewpoints that can apply in a given context, and must therefore not be suppressed in the computer-assisted environment. Extensively analyzed, argued, and negotiated, differing viewpoints lead to a more comprehensive understanding of the problem and presumably a higher quality solution.

Monitor **Agent:** The presence of object-agents in a computer-based decision-support system will greatly increase the number of possible and likely collaborative interactions. In fact, non-convergence (i.e., the inability of the agents to come to a consensus) is a real possibility. In the FEAT, ICODES and CIAT applications this potential problem could be controlled through various techniques, such as user interaction and the assignment of priorities (Pohl et al.1989). In the KOALA system the problem of non-convergence is much more serious, not only because of the relatively large number of object-agents but also because of the different viewpoints that these agents represent. For this reason the concept of ***Monitor*** agents has been introduced in the KOALA system. Essentially acting as facilitators, it is the task of ***Monitor*** agents to identify possible conflicts and assist in their resolution. They perform this responsibility through the application of moderating techniques that have been successful in human collaborations (Smith 1994, Cawsey 1992).

9.2.1 Agent Communication

To assist in the realization of desired outcomes and interests, each agent in KOALA is provided with the ability to communicate with other agents. Fundamental to collaborative decision-support, this communication is primarily used as a vehicle whereby agents attempt to satisfy their constraints and promote their interests. Theoretically, these agents would not necessarily be restricted to the local system environment. An agent has the potential of collaborating with any other agent located on a connected system. This is true provided both share a common language, or vocabulary. A language in this sense can be defined as a collection of syntax together with a set of associated semantics. Together, these components allow for the communication of concepts, ideas, and desires.

For example, let us assume that an agent requires a certain resource that is not readily available within the local environment. The agent may choose to broadcast a request for assistance on an open communication channel. Any agent capable of providing such a resource, or knowing another agent that does, may send a response back to the requesting agent. One or more of these agents may then be contacted by the initial agent in an effort to obtain the resource. It should be noted that all of this activity may take place transparent to the user's knowledge. The agent effectively is able to *take the initiative*, on the assumption that the requesting agent and the agent providing the resource share a common vocabulary. However, as in real life this is not always the case. In future more

mature versions of the KOALA system new languages could actually be learned by the agents. Similar to the technique employed by an infant, a basic understanding of a language could be acquired through observation (Burger 1993). An agent could monitor conversations engaged in by the target agent. Through observation, the agent would attempt to associate perceived semantics with the spoken syntax. Once a fundamental understanding of the language has been obtained, the agent could potentially enter into a collaborative dialog within the new arena.

9.2.2 Agent Collaboration

When a design action occurs in KOALA, each affected ***Space*** agent formulates a supporting set of design decisions based on individual constraints and interests. These decisions may include, for example, a new building material or structural system, and are presented to the other agents with the intent of achieving global acceptance. As a result, each agent gains exposure to various alternative solutions. However, due to their autonomous nature, ***Space*** agents will tend to lobby only for outcomes that best satisfy their particular interests. In other words, if left to their own devices, ***Space*** agents are reluctant to accept anything that offers a less than perfect outcome as determined from their perspective.

Upon receiving an alternative suggestion, an agent performs an analysis to determine the impact on itself of accepting such a design decision. This process may include numerous consultations with various service-agents pertaining to any domain specific analysis. Based on the result, the suggestion is then prioritized into a list of alternative solutions kept by each agent. The prioritization is grounded on how favorable the resulting outcome is for that particular agent. In other words, how close is the outcome to providing total (i.e., 100%) satisfaction? If the calculated percentage is high enough, the agent may accept the suggestion and proceed to add it to its set of acceptable solutions. Otherwise, the agent may choose to modify the suggestion to yield a more acceptable outcome for itself and present it to the other agents as an alternative solution. In either case, the receiving agent indicates its degree of acceptance of the proposed solution to the other agents. If at any time during the course of this deliberation the intersection of each agent's set of acceptable solutions yields a non-empty set, global consensus has been achieved. Otherwise, collaboration will continue in this manner until such a state is reached or a third party mediator intervenes.

9.2.3 Moderating Techniques and Strategies

As implied by their name, ***Monitor*** agents spend much of their time monitoring the interactions among ***Space*** agents. During the course of *listening in* on agent conversations, ***Monitor*** agents look for behavior that suggests real or potential conflict, and attempt to provide a resolution. This is a difficult undertaking requiring a great deal of research beyond the scope of the current implementation of KOALA. However, some relatively simple conflict detection and resolution strategies are being implemented in KOALA to demonstrate the feasibility of the ***Monitor*** agent concept. Three such strategies, namely

the *persuasive* strategy, the *imposive* strategy, and the *user-directed* strategy, are briefly described below.

Persuasive* Strategy:** As the name implies, this conflict resolution strategy attempts to use persuasion as a means of achieving global consensus. In essence, the ***Monitor agent attempts to persuade one or more agents to reevaluate previously unacceptable solutions based on a more flexible heuristic. The disadvantage of this approach is the additional expense accrued by requiring an extensive reevaluation of each proposed solution. A less costly approach would be to simply have each agent lower its minimum level of acceptability thus permitting more solutions to fall into an acceptable range. However, such a superficial approach would result in poorly thought out and potentially inadequate solutions and is therefore unacceptable.

As reevaluation proceeds, an agent may accept a previously unacceptable solution. In this case, the agent communicates the new decision to the ***Monitor*** agent along with an indication of its desirability. This desirability is based on the degree of loss the agent would be required to endure by accepting such a solution. Upon receiving an agent's decision regarding a reevaluated solution, the ***Monitor*** agent places the response along with a description of the particular solution into a list. Once the reevaluation task has been completed, the ***Monitor*** agent reviews this list searching for commonality. If a common solution is found, the ***Monitor*** agent indicates the selected solution to the agents. If the agents have found common agreement with multiple solutions, the ***Monitor*** agent uses the associated desirability as the decisive factor in determining which solution results in the least amount of penalty to the agent. However, at any time an agent may lodge a formal protest as an appeal against the consensus agreement.

Imposive* Strategy:** Employing a more forceful approach, *imposive* conflict resolution again attempts to bring about a global consensus through compromise. With this approach, the ***Monitor agent attempts to impose a solution onto the agents considering that persuasion proved to be ineffective. However, the imposed solution is by no means arbitrary. Rather, the solution is not only a product of agent collaboration, but it may actually be held favorably by a number of agents. In determining which solution to select, the mediating ***Monitor*** agent searches for a majority opinion. For example, suppose that three out of nine agents find acceptability with a certain solution 'A'. Further, suppose that of the remaining six agents no more than two agree on any one solution. Therefore, solution 'A' would attain a majority status. In this case, the ***Monitor*** agent would strongly consider imposing solution 'A' on all nine agents depending on the degrees of loss. Again, any agent displeased with the decision would be free to express its dissatisfaction via a formal protest.

While being somewhat dictatorial in nature, *imposive* conflict resolution does attempt to provide a solution that is desirable to the majority of agents. Even so, *imposive* conflict resolution is not without its limitations. There are circumstances under which this conflict resolution strategy cannot be successfully applied. Since the *imposive* strategy relies on the existence of a majority solution, it is certainly possible that none of the agents find another agent's solution acceptable. In this case there would be no majority solution for the ***Monitor*** agent to impose. Rather than have the ***Monitor*** agent arbitrarily select a

solution from among the agent suggestions, the ***Monitor*** agent is designed to employ a third strategy to resolve the conflict.

User-Directed **Strategy:** If both the *persuasive* and the *imposive* conflict resolution strategies have been unsuccessful in resolving the conflict, a more drastic approach is employed by the ***Monitor*** agent. As the name implies, *user-directed* conflict resolution involves the human designer as the definitive mediator. The ***Monitor*** agent initiates a dialog with the human designer presenting the particular dilemma at hand. In doing so, the ***Monitor*** agent provides the user with a description of the various solutions as presented by the deliberating agents. Such descriptions include the proposed solution, the agent that is presenting the solution, an indication of the major consequences of adopting that solution, and its overall desirability among the other agents. Based on this information, it is the task of the designer to decide on the most appropriate solution. However, the user is by no means confined to the solutions proposed by the agents, but is free to explore any number of alternative solutions or even postpone the decision to a later date. In the case of postponement the design continues despite the outstanding conflict.

Within such a collaboration intensive environment, outstanding conflicts may finally resolve themselves through future deliberation whether it involves the human designer or not. In any case, to assist in formulating a decision, the designer may choose to involve a number of agents in a hypothetical discussion of various alternatives. Similar to a ***Space*** agent engaging the assistance of a ***Structural*** domain agent to determine an appropriate structural system, the human designer may explore various consequences and alternatives via agent collaboration.

9.2.4 Formal Agent Protests

As part of its basic functionality, each ***Space*** agent has the ability to post formal protests or indicate domain violations. As with collaboration, *agent protest* is yet another method an agent can use to express its interests. This is particularly useful when an agent has been coerced by a ***Monitor*** agent into agreeing with a particular design decision. As a result, the agent may be dissatisfied with the outcome and now has the opportunity to express itself accordingly. This is accomplished via a formal protest procedure. An agent indicates its dissatisfaction by highlighting the border of its associated space in red. At any point during the design, the user may obtain a protest report by selecting the space. This report describes in detail the nature of the protest in addition to recommendations for its resolution.

Based on these recommendations, the user may choose to enter into a collaborative dialog with a collection of agents to pursue another course of action. Alternatively, the user may simply wish to view the agent's grievances making no attempt to resolve them at that particular time. Agent protest reports may be reviewed by the user at any time during the design activity. In any case, these grievances may resolve themselves through future collaboration or continue throughout the design activity. The fact that an agent is temporarily dissatisfied with the current solution does not prohibit the designer from progressing with a solution.

An extension to the method of *agent protest* described above would be to integrate a more entrepreneurial element into the notion of an agent. In other words, allow a discontent agent to attempt to perform some degree of retroactive negotiation with other agents in an effort to have the initial decision reevaluated in its favor. An agent may even bargain with other agents compromising its position on one issue in an effort to gain favor with another. However, it is not difficult to imagine the ramifications of allowing agents to essentially build alliances with other agents. This would clearly threaten to introduce a degree of bias into the decision-support environment that could very well result in a redefinition of agent interests and motivations.

9.3 System Description and Architecture

Like FEAT, ICODES and CIAT, KOALA is also an implementation of the ICDM development framework. Its principal components include a semantic network, a Graphical User Interface (GUI) world, and an agent world (Figure 75).

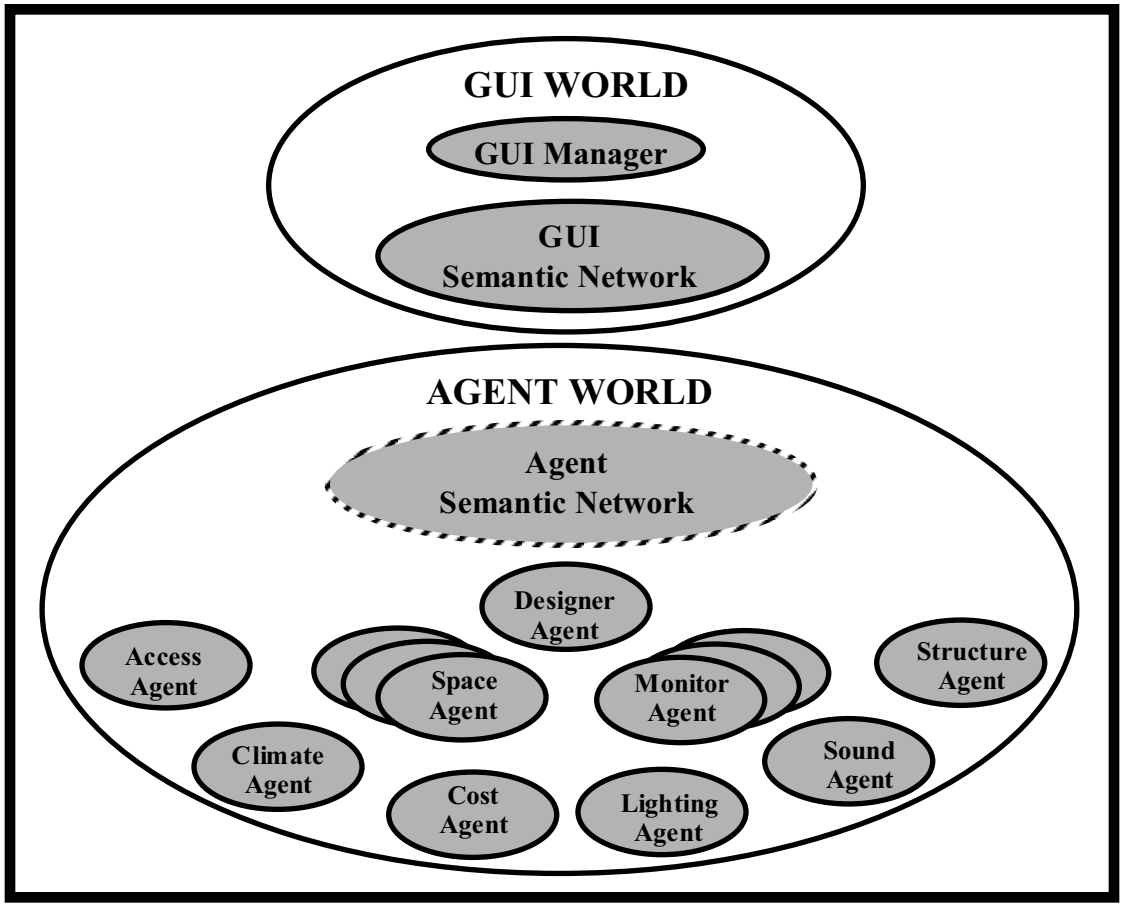


Figure 75: KOALA System Architecture

9.3.1 The Semantic Network

The semantic network exists as a structured collection of informational components that describe the design problem and the current solution at a given point in time. KOALA incorporates an object-oriented semantic network design (Papurt 1995) in which related attributes and functionality are encapsulated into object classes. In an object-oriented semantic network, knowledge components and their relationships are represented as objects. Applications can be broken down into a series of object classes according to attributes, functionality, and relationships to other objects. Class objects can be created and manipulated through function calls described by the class. In an abstract sense, these function calls can be thought of as messages that are sent to an object instructing it to take some kind of action.

The object-based semantic network employed in KOALA is divided into two components. The first component contains information relating to the GUI portion of the system describing the graphical attributes of the solution space including site boundaries and actual space objects. This information is used primarily to address the graphical aspects of the design environment. The second component of the semantic network relates to the logical aspects of the system. This includes design specifications, individual space characteristics, and the relationships describing their interactions. For efficiency, the majority of the semantic network is distributed across the ***Designer***, ***Service*** and ***Space*** agents.

The ***Designer*** agent contains semantic network objects describing the overall intent and objectives of the human designer. Service-agents contain the domain specific components of the semantic network. ***Space*** agents contain space dependent information.

For example, the ***Daylight*** agent has knowledge of the methods available for predicting the internal lighting characteristics of a space under given external daylight conditions. The ***Space*** agent, on the other hand, has knowledge of its desired daylight environment. Despite this distribution, the semantic network remains a self-managing entity. Utilizing an object-oriented model, information can be obtained or manipulated by any agent via the appropriate functional interface.

9.3.2 The GUI Manager

The GUI manager is responsible for the management of the graphical interface of the system. This encompasses menus allowing the user to initiate various actions, dialog boxes, which are used to present information to the user, and the graphical portion of the semantic network. Implemented in the C++ language (Stroustrup 1987), the components are formally defined in object classes. For example, a dialog box used by ***Space*** agents to express discontent with a particular design decision can be fully described and managed in a single object class. This is due to the fact that at the user-interface level all protest reports provide similar functionality and structure. Therefore, these characteristics can be encapsulated into a single *Protest Report GUI* object class. As a result, the GUI manager comprises a set of such self-contained, self-managing object classes.

To perform actual user-interface manipulations, the GUI manager employs a lower level library of tools (i.e., UIToolBox) developed in the CADRC as an object-oriented front-end to the Motif and X-Window user-interface and event management programming environment (Young 1990).

9.3.3 The Agent Manager

The task of the Agent manager is to manage the entire taxonomy of agents described in Section 9.2. To facilitate this task, all agents reside in a single environment referred to as the a*gent world*. Apart from its agent inhabitants, this world also contains the logical component of the semantic network. Taking advantage of its opportunistic nature, the *agent world* is implemented in the CLIPS-COOL language (NASA 1992). Providing an object-oriented approach to rule-based paradigms, COOL allows for object functionality to exist at both a procedural (e.g., message-handler or function call) and an opportunistic (e.g., rule firing) level. Further, COOL supports the division of applications into separate replicated environments known as modules. Each module contains its own knowledge and event management environment.

Using such a facility, applications can essentially encapsulate related knowledge and functionality into separate, self-contained modules. Information is passed between modules via an import/export mechanism. Scheduling of modules is performed by the Distributor coordination facility developed in the CADRC. Employing a "round-robin" approach, the Distributor provides each module the potential of executing its rules. Once scheduled, a module fires rules until a predetermined rule limit is reached or its agenda has been exhausted. In either case, the Distributor then passes control to the next module in the queue. This pattern repeats itself until there are no more rules to fire, at which time the *agent world* rests waiting for the next activity to occur. Taking advantage of this functionality, agents of a more static nature can be defined and implemented as modules. More dynamic agents, however, are implemented with greater efficiency as COOL objects.

Designer* Agent:** Due to its static nature the ***Designer agent is best implemented as a single CLIPS module. Having access to the interface-generating services of the GUI manager, the ***Designer*** agent may communicate with the human designer at any time during the evolving design. Such interaction may include verifying internal assumptions or even requesting additional insight into user intent and objectives. Being in continuous communication with both the user and the other agents, the ***Designer*** agent is essentially responsible for keeping the agents focused on the intent and objectives of the human designer.

Service-Agents: Also static in nature, service-agents are best described as CLIPS modules. Each service-agent resides in its own module. These modules comprise domain specific rules that are driven by requests from other agents. To employ the services of a service-agent the requesting agent creates a *request* object, which describes the nature of the desired service. Once the request has been broadcast by the requester, the appropriate

service-agent(s) take action. Processing of a request may require additional information to be obtained from other semantic network objects. Further, servicing of a request may require the employment of additional agents. Once processing has begun, service-agents are capable of obtaining or employing any resources or services they require to satisfy the request, without the assistance of the requester.

In the initial version of KOALA, a group of six architectural domains was selected. These domains are represented by ***Access***, ***Climate***, ***Sound***, ***Daylight***, ***Structure***, and ***Cost*** agents.

Space **Agents:** More dynamic in nature, ***Space*** agents are defined as COOL objects. Similar to the *Space* class, the basic attributes and functionality of an agent are described in an *Agent* base-class (Papurt 1995). Utilizing their definitions as a basis, the ***Space*** agent class is derived by inheriting the characteristics of both the *Space* and *Agent* base-classes. Additional characteristics describing specific interests and desires are then added to complete the definition. A ***Space*** agent object is instantiated with the addition of each new space into the progressing design. Once instantiated, the ***Space*** agent is free to interact with the other agents as desired. If a space is removed from the current state of the design solution, its associated ***Space*** agent is automatically destroyed.

Monitor **Agents:** Similar to ***Space*** agents, ***Monitor*** agents are dynamic in nature. As the number of spaces increases, so may the amount of agent collaboration. This increased collaboration may take place as several separate conversations occurring in parallel. To monitor these conversations additional ***Monitor*** agents may be required. With this in mind, ***Monitor*** agents are defined as COOL objects. Like the ***Space*** agent class, the ***Monitor*** agent class inherits its fundamental attributes and functionality from the *Agent* base-class. As the amount of agent collaboration fluctuates, so does the ***Monitor*** agent population.

9.3.4 The Agent Status Display Manager

The Agent Status Display (ASD) manager is an extension of the GUI manager. Like the GUI manager, the ASD manager employs the services of the UIToolBox library to generate its graphical user-interface components. In essence, the ASD manager provides a general user-interface for the agent population within the system. Each type of agent is represented as an active icon within a status menu. Service-agents have their own individual entries in this menu. However, due to their dynamic nature both the ***Space*** agent and ***Monitor*** agent populations are represented by a single pair of icon buttons.

The purpose of this active icon menu is twofold. Agents can indicate their current status through manipulating a color-coded border surrounding their icon button. For example, agent collaboration is illustrated by each agent turning its icon border yellow when it is in an active state. In the case of more dynamic agents such as ***Space*** agents, agent status is indicated more specifically through direct manipulation of the agent's GUI. For example, a ***Space*** agent can indicate its dissatisfaction with a particular design decision by turning the border of its graphical space representation red in addition to the border color change

of the ***Space*** agent's icon in the ASD menu. Whether in a general or specific form, the icon menu provided by the ASD manager is useful in providing agent expression in a graphical manner.

The second function of the ASD manager is to provide a general entry point for the user to address individual service-agents. This need may arise during the formation of a hypothetical consultation committee. The user may click on an agent's icon button as a means of directly addressing that agent or agent population. Once such an activity has been detected, the ASD manager conveys this information to the Agent manager, which in turn notifies the appropriate agent(s). Once notified, the agent interacts with the user employing the services of the GUI manager to construct an appropriate user-interface.

9.4 Interactions During a KOALA Session

A detailed description of the typical discourse between user and agents during a KOALA design session can be found elsewhere (Pohl 1996). This description, which is too long to include here, clearly demonstrates the parallel nature, potential for local decision-making, and decentralized communication in a decision-support system that includes object-agents. It suffices here to present a short excerpt.

For the purpose of this brief demonstration scenario excerpt it should be assumed that KOALA has been initialized with several categories of information, such as general project information, building type prototype information, and project specific criteria. Each of these categories is based on some general specifications for space type and occupant activity characteristics. The demonstration scenario focuses on the design of the 'Shoreline Community Center' in Pismo Beach, California. Data pertaining to bounding traffic ways, noise sources, climate, construction costs, prototypical building type information, and site information can be freely accessed by any member of KOALA's agent population.

To emphasize the relationship between agent collaboration and the kind of collaboration that occurs in human society, Pohl (1996) presents the entire demonstration scenario as a theatrical play. The setting is a site located in Pismo Beach and its surrounding characteristics. The playbill includes a list of character roles that are performed by the various agents in conjunction with the human designer. In keeping with this theatrical theme, interaction among the performers is presented in dialog form. Although KOALA agents have no emotional capabilities, given the theatrical nature of this example, agent dialog is presented in an animated and colorful manner.

Based on the information gathered from the prototype databases, KOALA presents the user with a set of spaces typical to a community center, such as 'lobby', 'reception', 'office', 'conference room', 'library', etc. (Figure 76). These space templates already contain a rich collection of prototypical characteristics and qualities such as area, desired orientation, and desired adjacency. The designer may select, or instantiate, any number of

these space types or may choose to define a new one. The following excerpt from Pohl (1996) focuses on the behavior of an *assembly* space (i.e., a 'lobby') after it has been selected by the user to become part of the evolving building design solution.

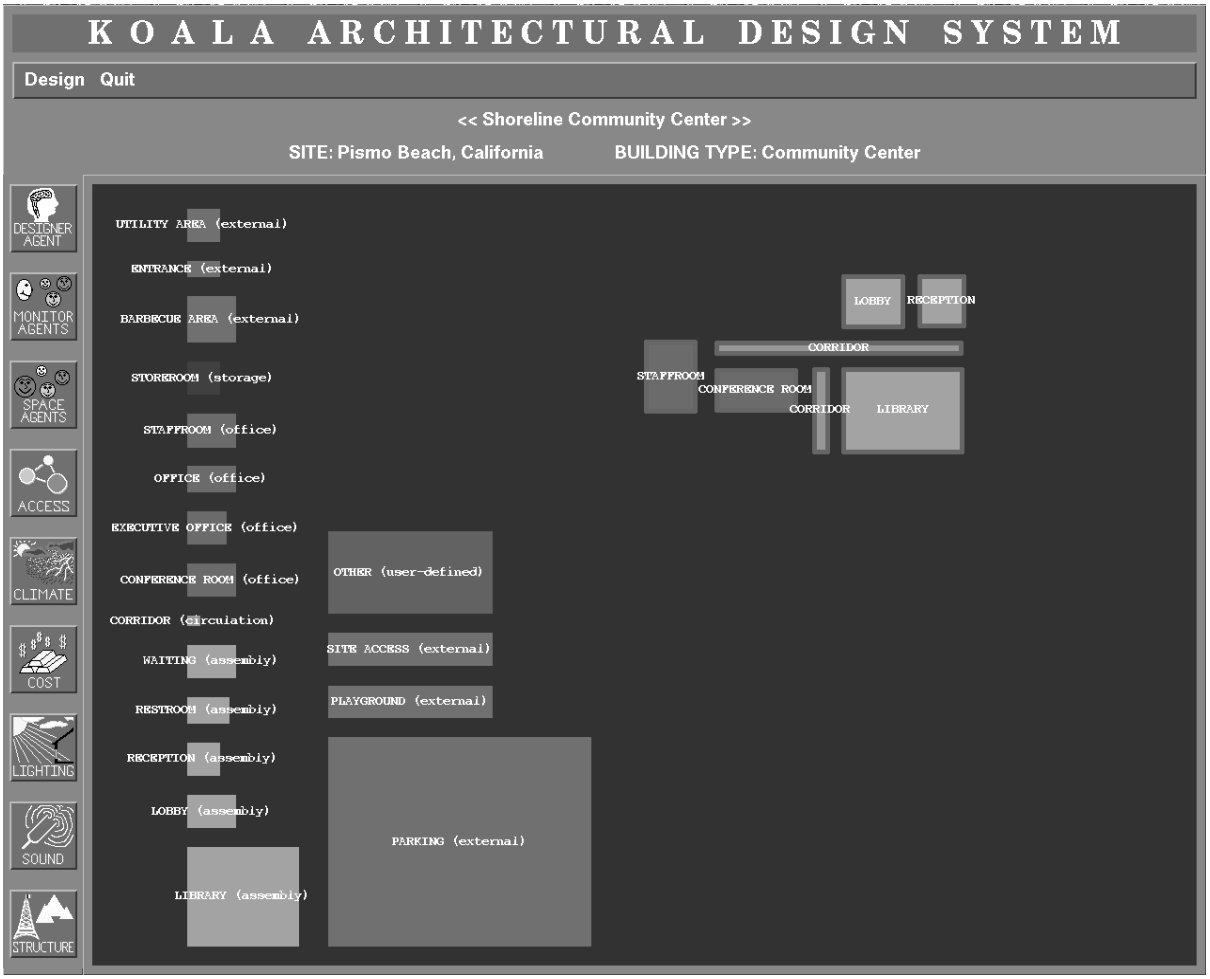


Figure 76: Main KOALA User Interface Screen

USER : *Instantiates and places a new 'assembly' space with the following characteristics:*

(Note that many of these characteristics are initialized with prototypical values. However, the user is free to redefine these values as desired.)

Space Type: assembly
Name: LOBBY
Area (Sq. Ft.): 270
# of Occupants: 10
Ceiling Height (Ft.): 8
Dimensions (Ft.): 15 x 18

Adjacencies:	**Necessary**	**Desirable**	**Optional**	**Undesirable**
	library	*conference*	*manager*	*staffroom*
	corridor	*parking*	*office*	*storeroom*
	reception		*barbecue area*	*utility area*
	restroom		*playground*	
	waiting			
	bldg. entrance			
	site access			

Activities: ***Code for when activities occur (D = Day Only; N = Night Only; A = Day and Night)***

Necessary	**Desirable**	**Optional**	**Undesirable**
reading(A)	*conversing(A)*	*conferencing(A)*	*drinking(A)*
telephoning(A)	*listening(A)*	*interviewing(D)*	*eating(A)*
viewing(A)	*resting(A)*		*storing(A)*
	writing(A)		*washing(A)*

---

*(Having just entered the solution space, LOBBY takes the initiative and attempts to educate itself as to its location and immediate surroundings. LOBBY does this by posing questions to other agents. As a **Space** agent, LOBBY has the ability to engage in an interactive dialog with any member of the agent population including the human designer agent. While some of these questions can be directed toward a certain agent specifically, others are posed to the general agent population as a whole.)*

*(The first order of business for LOBBY is to announce its arrival as a member of the evolving solution. This takes the form of an open declaration to all other **Space** agents. Upon receiving such an announcement, a **Space** agent will in turn send back a reply indicating its existence in the design world)*

LOBBY : "Hello. My name is LOBBY and I have just entered the solution space. Who else is out there?"

(While awaiting responses from any other spaces that may exist, LOBBY also begins the process of trying to satisfy its needs and desires. To perform this task, LOBBY has at its disposal a robust collection of expert consultants in the form of service-agents.)

(The set of needs and desires first addressed by LOBBY are related to lighting. The first step in this task is to determine the degree to which the designer wishes to involve daylight as opposed to artificial lighting.)

LOBBY : "What are the daylight utilization importance and percentages for desired Task Illumination by Daylight (TID), and Background Illumination by Daylight (BID)?"

(Responding to requests of this nature, the Designer agent replies to LOBBY conveying the user's intentions with respect to daylight utilization.)

DESIGNER AGENT : "LOBBY, daylight utilization is *necessary* and the desired TID is greater than 50% and BID is greater than 90%."

(Based on the desired TID and BID, LOBBY then attempts to determine exactly how much daylight must enter its space to satisfy its illumination requirements.)

LOBBY : *Determines maximum daylight task illumination (DTI) and maximum daylight background illumination (DBI) as follows:*

DTI = (TID/100)(highest task illum. among activities occurring in the space)
= (50.0/100)(60 FC)
= 30 FC

DBI = (BID/100)(highest background illum. among activities occurring in the space)
= (90.0/100)(30 FC)
= 27 FC

(Once the maximum DTI and DBI have been determined, LOBBY inquires as to the amount of window area it would require to obtain its desired lighting levels.)

LOBBY : "What is the minimum window area required in my shortest wall to satisfy a DTI requirement of 30 FC?" (The relevant characteristics of the selected wall within its space environment accompany the request).

*(Fielding this request, the **Daylight** agent employs the services of another agent to assist in formulating a response.)*

DAYLIGHT : "What is the average external illumination level for the climate of Pismo Beach, CA.?"

CLIMATE : "DAYLIGHT, the average external illumination level for Pismo Beach, CA is 1250 FC."

*(In formulating its recommendations, **DAYLIGHT** makes several assumptions including: a reference point located in the center of the space*

at an elevation of 3 ft.; an Internally Reflected Component of Daylight Factor equal to 15% of the Sky Component; transmission loss due to glass, window frame, and dirt to be 20% of the Sky Component; and no external obstructions that would produce an Externally Reflected Component. In addition, windows are assumed to have a sill height of 3 ft. and continue to the ceiling. To satisfy LOBBY's request, ***DAYLIGHT*** *employs several formulas to determine an appropriate window area. After applying these formulas,* ***DAYLIGHT*** *determines that a DTI requirement of 30 FC. would be satisfied by a window having a width of approximately 20 ft. However, the wall which LOBBY selected to contain this window is only 15 ft. wide. With this in mind,* ***DAYLIGHT*** *goes one step further and calculates the illumination which LOBBY could obtain by devoting the entire 15 ft. span to a window wall.)*

DAYLIGHT : "LOBBY, based on your specifications you would need to have a window with a width of 20 ft. to provide 30 FC. of illumination. However, considering that your selected wall has a span of only 15 ft., you could obtain 17.5 FC. of illumination if you devoted the entire wall to window area."

(Receiving ***DAYLIGHT's*** *recommendations, LOBBY soon realizes its predicament. To obtain at least 50% of its task illumination from daylight requires a window 20 ft. wide. The illumination resulting from a 15 ft. window under the current configuration is only 17.5 FC. Unwilling to accept this value, LOBBY determines another course of action. LOBBY repeats its request for the daylight analysis but this time focusing on its larger wal,l which has a span of 18 ft.)*

LOBBY : "What is the minimum window area required in my largest wall to satisfy a DTI requirement of 30 FC?" *Assume that the relevant characteristics of the selected wall within its space environment accompany the request.*

DAYLIGHT : "LOBBY, based on your specifications you would need to have a window with a width of 12 ft. to provide 30 FC of illumination."

(Satisfied with this result, LOBBY accepts ***DAYLIGHT's*** *recommendation and modifies itself so that the selected wall now has an external orientation and reflects the insertion of a 12 ft. wide window. It should be noted that LOBBY's decision regarding the alignment of the wall containing the window only goes as far as deciding on internal or external orientation. Taking into account the direction-independent quality of daylight with respect to illumination, deciding on a geographic orientation is somewhat premature and is, therefore, not a concern at this point.)*

(As an aside, if LOBBY's current configuration did not, for example, allow for the placement of an adequately sized window in the target wall then LOBBY would still have a number of alternative courses of action. These

alternatives range from simply accepting a lower illumination level, but flagging a daylight domain violation, to requesting authorization from the user to resize its shorter wall to now allow for the insertion of a 20 ft. wide window thus providing the desired daylight illumination level within the space.)

The scenario continues with the introduction of additional Space agents and concurrent consideration of multiple domains, such as noise control, energy conservation through the adoption of passive solar design principles, and so on.

10. References and Bibliography

Aitchison J. and J.Brown (1957); 'The Log-Normal Distribution'; Cambridge University Press, Cambridge, England.

Alexander B. (1993); 'How Great Generals Win'; Avon Books, New York, New York.

Archea J. (1987); 'Puzzle-Making: What Architects Do When No One Is Looking'; in Kalay (ed.) Principles of Computer-Aided Design: Computability of Design'; Wiley, New York, New York.

Autodesk (1993); 'AutoCAD and 3-D Studio Software'; Autodesk Inc., Sausalito, California.

Bairstow J. (1987); 'Personal Workstations Redefine Desktop Computing'; High Technology, March (pp.18-23).

Beguelin A, J. Dongarra, G. Geist, R. Manchek and V. Sunderam (1991); 'A User's Guide to PVM Parallel Virtual Machine'; Technical Report TM-11826, Oak Ridge National Laboratory, Oak Ridge, Tennessee.

Bergman (1996); 'A Ticket to Ride the Dragon'; Marine Corps Gazette, 80(2), February (pp.12-13).

Birman K. and K. Marzullo (1989); 'ISIS and the META Project'; Sun Technology, Summer (pp.90-104).

Bohm D. (1983); 'Wholeness and the Implicate Order'; Ark Paperbacks, London, England.

Brooks R.A. (1990); 'Elephants Don't Play Chess'; in Maes P. (ed.) Designing Autonomous Agents, MIT Press, Cambridge, Massachusetts (pp.3-15).

Buchanan B. and E. Shortliffe (1984); 'Uncertainty and Evidential Support'; in Buchanan and Shortliffe (eds.) Rule-Based Expert Systems, Addison-Wesley, Reading, Massachusetts (pp.209-232).

Bureau of Explosives (1992); 'Hazardous Materials Regulations of the Department of Transportation by Air, Rail, Highway and Water including Specifications for Shipping Containers'; Tariff No. BOE-6000-L, Washington, DC.

Burger J. (1993); 'Personality', Brooks/Cole Publishing Co., Pacific Grove, California, (pp.409-411, 471-475)

CADRC (1994); 'ICODES: Proof-of-Concept System: Final Report'; Contract #: N47408-93-7347, Naval Civil Engineering Laboratory (Port Hueneme, California), CAD Research Center, Cal Poly, San Luis Obispo, California.

Cawsey A. (1992); 'Explanation and Interaction', MIT Press, Cambridge, Massachusetts.

Chaib-Draa B., R. Mandiau and P. Millot (1992); 'Distributed Artificial Intelligence: An Annotated Bibliography'; Sigart Bulletin, ACM Press, 3(3), August.

Clausewitz von C. (1976); 'On War'; Princeton University Press, Princeton, New Jersey.

Creveld van M. (1986); 'Command in War'; Harvard University Press, Cambridge, Massachusetts.

Conrey S., R. Meyer and V. Lesser (1988); 'Multistage Negotiation in Distributed Planning'; in Bond and Gasser (eds.) Readings in Distributed Artificial Intelligence, Morgan Kaufmann, Los Altos, California (pp.367-386).

Davis R. and R. Smith (1983); 'Negotiation as Metaphor for Distributed Problem Solving'; Artificial Intelligence, Vol.20, Jan. (pp.63-109).

Dawkins R. (1987); 'The Blind Watchmaker: Why the Evidence of Evolution Reveals a Universe Without Design'; Norton and Co., New York, New York.

Dejong G. (1982); 'An Overview of the Frump System'; Lehnert and Ringle (eds.) Strategies for Natural Language Processing, Lawrence Erlbaum, Hillsdale, New Jersey (pp.149-176).

Dreyfus H. and S. Dreyfus (1988); 'Making a Mind Versus Modeling a Brain: Artificial Intelligence Back at a Branchpoint'; Daedalus, 117, Winter (pp.15-43).

Duda R., P. Hart, N. Nilsson, R. Reboh, J. Slocum and G. Sutherland (1977); 'Development of a Computer-Based Consultant for Mineral Exploration'; SRI Report, Stanford Research Institute, Menlo Park, California, October.

Durfee E. (1988); 'Coordination of Distributed Problem Solvers'; Kluwer Academic, Boston, Massachusetts.

Durfee E. and T.Montgomery (1990); 'A Hierarchical Protocol for Coordination of Multiagent Behavior'; Proc. 8th National Conference on Artificial Intelligence, Boston, Massachusetts (pp.86-93).

Ellis C. (1989); 'Explanation in Intelligent Systems'; in Ellis (ed.) Expert Knowledge and Explanation: The Knowledge-Language Interface, Horwood, England.

Elmasri R. and S.B.Navathe (1989); 'Fundamentals of Database Systems'; Benjamin-Cummings, Redwood City, California.

Fischer G. and K. Nakakoji (1991); 'Making Design Objects Relevant to the Task at Hand'; Proc. AAAI-91 Ninth National Conference on Artificial Intelligence, MIT Press, Cambridge, Massachusetts (pp.67-73).
Forsyth R. (1989); 'Machine Learning: Principles and Techniques'; Chapman and Hall, Computing Series, London, England.

Frensch P. and J. Funke (1995); 'Complex Problem Solving: The European Perspective'; Erlbaum, Hillsdale, New York.

Fu K. S. and T. L. Booth (1975); 'Grammatical Inference: Introduction and Survey'; IEEE Transactions on Systems, Man, and Cybernetics. SMC-5: (pp.95-111, 409-423).

Gero J., M. Maher and W. Zhang (1988); 'Chunking Structural Design Knowledge as Prototypes'; Working Paper, The Architectural Computing Unit, Department of Architectural and Design Science, University of Sydney, Sydney, Australia.

Gregory K. (1992); 'Programming with Motif'; Springer Verlag, New York, New York.

Hayden H. (1995); 'Warfighting: Maneuver Warfare in the US Marine Corps'; Stackpole Books, Mechanicsburg, Pennsylvania.

Hayes P. J. and S. P. Weinstein (1991); 'Construe-TIS: A System for Content-Based Indexing of a Database of News Stories'; Rappaport and Smith (eds.) Innovative Applications of Artificial Intelligence 2, AAAI Press, Menlo Park, California (pp.47-64).

Huberman B. A. (1991); 'The Performance of Cooperative Processes'; in Forrest S. (ed.) Emergent Computation, MIT Press, Cambridge, Massachusetts (pp.38-47).

Huberman B. A. and T. Hogg (1988); 'The Behavior of Computational Ecologies'; in Huberman B.A. (ed.) The Ecology of Computation, North-Holland, Amsterdam (pp.71-115).

Jacobs P. S., and L. F. Rau (1988); 'A Friendly Merger of Conceptual Analysis and Linguistic Processing in a Text Processing System'; Proceedings of the Fourth IEEE AI Applications Conference, IEEE Computer Society Press, Los Alamitos, California (pp.351-356).

Johnson-Laird P. (1993); 'Human and machine Thinking'; Erlbaum, Hillsdale, New Jersey.

KL Group (1995); 'XRT/Table for Motif: Programmer's Guide and Reference Manual'; KL Group (Toronto, Canada), November, New York, New York.

Kolodner J. (1993); 'Case-Based Reasoning'; Morgan Kaufmann, San Mateo, California.

Kraus S. and J. Wilkenfeld (1990); 'The Function of Time in Cooperative Negotiations'; in Huhns (ed.) Proc. 10th International Workshop on Distributed Artificial Intelligence, Bandera, Texas.

Krebs C. J. (1972); 'Ecology'; Harper and Row, New York, New York.

Krulak C. (1996); 'Embracing Innovation'; marine Corps Gazette, 80(1), Jan. (pp.18-20).

Kuhn T. (1977); 'The Essential Tension: Selected Studies in Scientific Tradition and Change'; University of Chicago Press, Chicago, Illinois.

Kurzweil R. (1992); 'The Age of Intelligent Machines'; MIT Press, Cambridge, Massachusetts.

Lakoff G. and M. Johnson (1980); 'Metaphors We Live By'; University of Chicago Press, Chicago, Illinois.

Larsi B., H. Larsi and V. Lesser (1990); 'Negotiation and its Role in Cooperative Distributed Problem Solving'; in Huhns (ed.) Proc. 10th International Workshop on Distributed Artificial Intelligence, Bandera, Texas.

Leonhard R. (1991); 'The Art of Maneuver'; Presidio Press, Novato, California.

Lesser V. (ed.) (1995); 'Proc. First International Conference on Multi-Agent Systems'; ICMAS-95, AAAI Press/MIT Press, Cambridge, Massachusetts.

Maturana H. R., J. Lettvin, W. McCulloch and W. Pitts (1960); 'Anatomy and Physiology of Vision in the Frog'; Journal of General Physiology, 43 (pp.129-175).

Michalski R. (1983); 'A Theory and Methodology of Inductive Learning'; Artificial Intelligence, Vol.20 (pp.111-161).

Mitchell T., J. Allen, P. Chalasani, J. Cheng, O. Etzioni, M. Ringuette and J. Schlimmer (1991); 'Theo: A Framework for Self-Improving Systems'; VanLehn (ed.) Architectures for Intelligence, Twenty-Second Carnegie Mellon Symposium on Cognition, Lawrence Erlbaum, Hillsdale, New Jersey (pp.323-355).

Montroll E. and M. Shlesinger (1982); 'On 1/f Noise and Other Distributions with Long Tails'; Proc. National Academy of Science US 79 (pp.3380-3).

Myers L., J. Pohl, J. Cotton, J. Snyder, K. Pohl, S. Chien, S. Aly and T. Rodriguez (1993); 'Object Representation and the ICADS-Kernel Design'; Technical Report CADRU-08-93, CAD Research Center, Design and Construction Institute, Cal Poly, San Luis Obispo, California, January.

Myers L. and K. Pohl (1994); 'Using PVM to Host CLIPS in Distributed Environments'; CLIPS Users Conference, NASA Johnson Space Center, Houston, Texas, Sep.12-14.

Nadendla R. and A. Davis (1995); 'FEAT: Distributed Problem Solving in a Military Mission Planning Environment'; Master Thesis, Computer Science Department, Cal Poly, San Luis Obispo, California.

NASA (1992); 'CLIPS Reference Manual', Software Technology Branch, Johnson Space Center, Houston, Texas.

NASA (1989); 'CLIPS Architecture Manual (Version 4.3)'; Artificial Intelligence Section, Johnson Space Center, Houston, Texas.

Pan J. and J. Tenenbaum (1991); 'Toward an Intelligent Agent Framework for Enterprise Integration'; Proc. Ninth National Conference on Artificial Intelligence, vol.1, San Diego, California, July 14-19 (pp.206-212).

Papurt D. (1995); 'Inside the Object Model', SIGS Books, New York, New York.

Patterson D. (1996); 'A Microcelebration'; Computer, October.

Pike R. et al. (1990); 'Plan 9 from Bell Labs'; Research Note, Bell Labs, July.

Pohl J. (1996); 'Agents and their Role in Computer-Based Decision Support Systems'; in Pohl J. (ed.) Advances in Cooperative Environmental Design Systems, focus symposium: International Conference on Systems Research, Informatics and Cybernetics, Baden-Baden, Germany, August 14-18 (pp.41-54).

Pohl J., L. Myers and A. Chapman (1994); 'Thoughts on the Evolution of Computer-Assisted Design'; Technical Report, CADRU-09-94, CAD Research Center, Design and Construction Institute, College of Architecture and Environmental Design, Cal Poly, San Luis Obispo, California, September.

Pohl J., J. La Porta, K. Pohl and J. Snyder (1992); 'AEDOT Prototype (1.1): An Implementation of the ICADS Model'; Technical Report, CADRU-07-92, CAD Research Center, Design and Construction Institute, College of Architecture and Environmental Design, Cal Poly, San Luis Obispo, California.

Pohl J., L. Myers, A. Chapman, J. Snyder, H. Chauvet, J. Cotton, C. Johnson and D. Johnson (1991); 'ICADS Working Model Version 2 and Future Directions'; Technical Report, CADRU-05-91, CAD Research Center, Design and Construction Institute, College of Architecture and Environmental Design, Cal Poly, San Luis Obispo, California.

Pohl J., L. Myers, A. Chapman and J. Cotton (1989); 'ICADS: Working Model Version 1'; Technical Report, CADRU-03-89, CAD Research Unit, Design and Construction Institute, College of Architecture and Environmental Design, Cal Poly, San Luis Obispo, California.

Pohl J., A. Chapman, L. Chirica, R. Howell and L. Myers (1988); 'Implementation Strategies for a Prototype ICADS Working Model'; Technical Report, CADRU-02-88, CAD Research Center, Design and Construction Institute, College of Architecture and Environmental Design, Cal Poly, San Luis Obispo, California.

Pohl K. (1996); 'KOALA: An Object-Agent Architectural Design System'; Master Thesis, Architecture Department; Cal Poly, San Luis Obispo, California.

Pohl K. (1996); 'KOALA: An Object-Agent Design System' in Pohl J. (ed.) Advances in Cooperative Environmental Design Systems, focus symposium: International Conference on Systems Research, Informatics and Cybernetics, Baden-Baden, Germany, August 14-18 (pp.81-92).

Pohl K. (1995); 'CMS: A PVM-Based Communication Facility for Cooperative Systems'; in Pohl J. (ed.) Advances in Cooperative Computer-Assisted Environmental Design Systems, Focus Symposium: 8th International Conference on Systems Research, Informatics and Cybernetics, Baden-Baden, Germany, August 16-20.

Reboh R. (1981); 'Knowledge Engineering Technologies and Tools in the Prospector Environment'; SRI Technical Note 243, Stanford Research Institute, Menlo Park, California, June.

Reitman W. (1964); 'Heuristic Decision Procedures, Open Constraints, and the Ill-Defined Problems'; in Shelley and Bryan (eds.) Human Judgments and Optimality, Wiley, New York, new York (pp.282-315).

Reitman W. (1965); 'Cognition and Thought'; Wiley, New York, New York.

Rittel H. and M. Webber (1984); 'Planning Problems are Wicked Problems'; in Cross (ed.) Developments in Design Methodology, Wiley, New York, New York (pp.135-144).

Rogue Wave (1992); 'Tools.h++: Introduction and Reference Manual'; Rogue Wave Software Inc., Corvallis, Oregon.

Rosenman M. and J. Gero (1993); 'Creativity in Design Using Design Prototypes', in Gero and Maher (eds.) Modeling Creativity and Knowledge-Based Creative Design, Erlbaum, Hillsdale, New Jersey (pp.111-138).

Sawyer R. (1995); 'Sun Pin: Military Methods'; Westview Press, Boulder, Colorado.

Schank R. and R. Osgood (1990); 'Content Theory of Memory Indexing'; Technical Report 2, The Institute for the Learning Sciences, Northwestern University.

Schank R. (1991); 'Case-Based Teaching: Four Experiences in Educational Sofware Design'; Technical Report 7, The Institute for the Learning Sciences, Northwestern University.

Scheifler R. and J. Gettys (1992); 'X-Window System (X Version 11 Release 5), Digital Press, Burlington, Massachusetts.

Schockley W. (1957); 'On the Statistics of Individual Variations of Productivity in Research Laboratories'; Proc. IRE 45 (pp.279-290).

Schmitt J. (1997); 'If Not Synchronization, What?'; Marine Corps Gazette, 81(1), January (pp.54-60).

Schmitt J. (1994); 'Mastering Tactics: A Tactical Decision Game Workbook'; marine Corps Association, Quantico, Virginia.

Schon D. (1983); 'The Reflective Practitioner: How Professionals Think in Action'; Basic Books.

Schon D. (1988); 'Designing: Rules, Types and Worlds'; Design Studies, 9(3), July (pp.181-190).

Senge P. (1993); 'Transforming the Practice of Management'; Human Resource Development Quarterly, Jossey-Bass, San Francisco, California.

Shapiro D. (1987); 'Structured Induction in Expert Systems'; Addison-Wesley, Reading, Massachusetts.

Simon H. (1984); 'The Structure of Ill-Structured Problems'; Cross (ed.) Developments in Design Methodology, Wiley, New York, New York (pp.145-166).

Simon H. (1981); 'The Sciences of the Artificial'; MIT Press, Cambridge, Massachusetts.

Slater R. (1989); 'Portraits in Silicon'; MIT Press, Cambridge, Massachusetts.

Smith J. (1994); 'Collective Intelligence in Computer-Based Collaboration', Erlbaum, Hillsdale, New Jersey.

Smith R. (1980); 'The Contract-Net Protocol: High-Level Communication and Control in a Distributed Problem Solver'; IEEE Transactions on Computers, C-29(12), December (pp.1104-13).

Stroustrup B. (1987); 'The C++ Programming Language'; Addison-Wesley, Reading, Massachusetts.

Sunderam V. (1990); 'PVM: A Framework for Parallel Distributed Computing'; Concurrency: Practice & Experience, 2(4), December.

Sycara K. (1989); 'Multiagent Compromise via Negotiation'; in Gasser and Huhn (eds.) Distributed Artificial Intelligence, Vol.II, Pitman, Los Altos, California (pp.119-138).

Taylor S. (1949); 'Science Past and Present'; Heinemann, London, England (pp.108-129).

Thornton C. (1992); 'Techniques in Computational Learning'; Chapman and Hall, Computing Series, London, England.

US Army (1994); 'USSOCOM C4I Strategy into the 21st Century'; US Special Operations Command, Office of the Commander in Chief (General W. Downing), McDill Air Force Base, Florida (pp.5).

US Army (1995); 'Force XXI: America's Army of the 21st Century'; Office of the Chief of Staff, Army, Ft. Monroe, Virginia: Louisiana Maneuvers Task Force, January 15th.

Wilson G., A. Cremarty, T. Adams, M. Grinberg, C. Tollander and J. Cunningham (1984); 'AI Assists Analysts in Identifying Soviet Radar Systems'; Defense Systems Review, January (pp.23-26).

Winograd T. and F. Flores (1987); 'Understanding Computers and Cognition'; Addison-Wesley, Reading, Massachusetts.

Wittgenstein L. (1953); 'Philosophical Investigations', Macmillan, New York, New York.

Young D. (1990); 'The X-Window System: Programming and Applications with Xt (OSF/Motif Edition)', Prentice-Hall, Englewood Cliffs, New Jersey.

Zinc (1993); 'ZINC Application Framework: Programmer's Reference'; Zinc Software Inc., Pleasant Grove, Utah.

11. Keyword Index

D

E

F

G

H

I

N

O

P