STELLINGEN

behorende bij het proefschrift

IMPROVING THE EFFECTIVENESS OF DECISION-MAKING USING A DISTRIBUTED DECISION SUPPORT SYSTEM

van

Theodor W. Schlickmann

19.10.1992
We have all heard the cry "get the user involved" in information system development. There should be rapid user/developer feedback on requirements, technical feasibility and cost. Both the systems implementation and the requirements could be closely managed. In reality, things that will happen when the user defines and the development community executes are:

- The user provides a set of requirements and then keeps changing them during development;
- The development community changes the requirements to make the system "better" for the user;
- The user develops requirements "throws them over the wall" to the developer and then the developer takes them as gospel.

II

Decision-making is heavily dependent upon information and knowledge that cannot be quantified or symbolized. Now, the best what Decision Support Systems can do for a decision maker is to provide him with the latest, best data in the clearest possible form and at his greatest convenience. However, we do not consider this to be a problem beyond solution, but a continuing challenge to both the decision maker and the developer.

III

Although the decision-making problems we are confronted with daily appear extremely varied at first glance, there is a common structure to which all decision-making problems can be reduced. Likewise there is a common basic structure for decision-making models, although these models may differ greatly in details.

IV

The decisions on options at various points in time are often interdependent, so that the current options cannot be decided upon in isolation from future actions. Therefore multi-stage (sequential) decision-making models are required for the explicit inclusion of intertemporal interdependencies. These decision-making models should plan future measures (more or less crudely) simultaneously with present measures.

V

Decision support systems come about for two reasons. One is the recognition or anticipation that decision efficiency needs to be enhanced; that there is a problem that needs correcting. The second reason is that a technological advantage is believed to exist for improving efficiency and giving a competitive edge. The first reason is often called as "requirements push," while the second is called as "technology pull".

VI

From an ergonomic point of view on DSS human limitations in decision-making are undoubtedly a significant aspect when developing a decision support system but they are in no way the first step of consideration. The main goal will be to optimize the effectiveness of the man-machine system to be designed or evaluated with respect to the mission to be fulfilled. Decision support systems will therefore differ considerable because of and with respect to their different missions, functions and tasks.

VII

The decision-making model cannot be understood without reference to the model of actual operating conditions (i.e., the Command and Control cycle). The decision support abilities of the model cannot be tested out of the context of a mission. The model's support characteristic ability is that it can detect and avoid or minimize problems in advance. In contrast, because of time pressure, the decision support model is poorly equipped to deal with ongoing catastrophe scenarios.

VIII

The longing for peace is part of the essential nature of normal human beings. The chance to live peaceful, ordered lives can be seen as a basic goal of human existence. "Peace is ordered tranquility" ("Pax est tranquillitas ordinis"), as Augustinus expressed it. But if this is taken out of context, problems become apparent. "The citizen's first duty is to remain calm" - even in times of turmoil? "We must bring order into the situation" - at the cost of peace? "Peace at any price" - at ANY price?

IX

The demonstrated economic disadvantages of arms buildups and economic benefits of arms reduction, and the availability of economically feasible transitional strategies, are politically powerless in the face of resistance to disarmament owing to security concerns.

X

Only when you leave a place and come back to it you can really understand or in the words of T.S. Eliots:

>> We shall not cease from exploration
And the end of all our exploring
Will be to arrive where we started
And know the place for the first time.<<

However, if we were able to enter the world of >the kangaroo< we would find the great revaluation not there, but upon our return to the >human world<.
IMPROVING THE EFFECTIVENESS OF DECISION-MAKING USING A DISTRIBUTED DECISION SUPPORT SYSTEM
Improving the effectiveness of decision-making using a distributed decision support system
IMPROVING THE EFFECTIVENESS OF DECISION-MAKING USING A DISTRIBUTED DECISION SUPPORT SYSTEM

proefschrift

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft
op gezag van de Rektor Magnificus
Prof. drs. P.A. Schenk,
in het openbaar te verdedigen
ten overstaan van een commissie
aangewezen door het College van Dekanen
op maandag 19 oktober 1992 te 14.00 uur

door

Theodor Wilhelm Schlickmann

geboren te Gelsenkirchen (Duitsland)
Dit proefschrift is goedgekeurd door de promotor:

Prof. Dr. H.G. Sol

en door de leden van de promotiecommissie:

Prof. Dr. E. Andriessen
Prof. Dr. A. F. Andrus
Dr. P. W. G. Bots
Prof. Dr. H. Koppelaar
Prof. Dr. M. Looijen
PREFACE

If life is hard for single individuals wrestling with their fate, then what happens in command and control systems, with interdependent decision-makers responsible for partially overlapping sections of complex problems? Addressing these situations is a logical next step for decision theory, but not one that it can take alone. Although the essential problem in command and control is still individuals pondering the unknown, there are now rigid machines, rules and doctrines in the picture, along with more fluid social relations. These require the skills of computer scientists, human-factor specialists, domain experts, and organizational theorists.

What follows is an attempt to pull these perspectives together into a framework for analyzing command and control systems. In doing so, we have characterized the problem more generally as distributed decision-making, defined as any situation in which decision-making information is not completely shared by those with a role in shaping the decision. The set of systems having this property includes high-tech examples, such as air-traffic control, and satellite management of a multinational corporation; mid-tech examples, such as forest fire-fighting and police dispatching; low-tech examples, such as a volunteer organization's coordination of its branches' activities, or a couple's integration of their childrearing practices or their use of a joint bank account - as well as the command of a military operation or a far-flung foreign service.

We have chosen to look far afield for examples, in the belief that it is possible to understand one's own situation better by looking at the circumstances of others. They may do things so well or so poorly as to set the viability of different strategies into sharp relief, as was the goal of 'In Search of Excellence' (Peters and Waterman [11]). Synthesizing the experience of diverse systems may highlight the significant dimensions in characterizing and designing other systems.
ACKNOWLEDGEMENTS

I am indebted to a number of people and organizations who helped me in the preparation of this study. First, I want to thank Henk Sol for the opportunity he has given me, for the many discussions we have had, and for the time he has invested in reading and re-reading the manuscript. I should also like to thank SHAPE Technical Centre for supporting my research and for providing a working environment that encourages creativity and technical excellence.

Second, I want to thank the many people who listened to my ideas and who were not afraid to ask hard questions. In particular, the following people influenced my thoughts in a number of dimensions: Pieter Bots, Wolfgang Gerok, Lars Ljungholm, Kurt Vanlehn, and Christian Schulz-Wolfgramm.

I appreciate the help also of the following reviewers: Loren Diedrichsen and Helmut Laux. Also, I want to thank my editor, Mike Baxter for his tireless encouragement and guidance.

Finally, I want to thank my wife, Anita, for her loving support during the life of this project.
# Table of Contents

1. INTRODUCTION................................................................. 1
   1.1. Defining the issue.................................................... 2
   1.2. Decision support and the military............................... 3
   1.3. Command and Control.............................................. 4
   1.4. The Key Tasks....................................................... 6
   1.5. Outline of study.................................................... 11

2. DEFINITIONS AND TECHNIQUES.......................................... 13
   2.1. Definitions from lattice theory................................... 13
   2.2. Logic programming.................................................. 15
   2.3. Logic modelling and decision-making........................... 21

3. EXPLORATION OF DECISION-MAKING...................................... 27
   3.1. Decision-making steps in relation to information systems... 27
   3.2. The role of information systems in decision-making........ 28
   3.3. Human factors in decision-making................................ 32
   3.4. Characteristics and constraints of human information-processing... 33
   3.5. Identifying decision-making constraints........................ 36
   3.6. Future possibilities................................................ 40

4. DEVELOPING A DISTRIBUTED DSS......................................... 43
   4.1. A methodology for development..................................... 43
   4.2. Modelling and design principles................................... 50
   4.3. Modelling distributed decision-making........................... 53

5. REQUIREMENTS ANALYSIS OF DECISION-MAKING PROCESSES.......... 59
   5.1. Introduction.......................................................... 59
   5.2. Main activities........................................................ 62
   5.3. Conclusions............................................................ 67
Table of Contents

6. LOGIC-BASED MODELLING OF DISTRIBUTED DECISION-MAKING ........ 69
   6.1. Model description ................................................. 69
   6.2. Coordination strategies for distributed systems ............. 71
   6.3. Algorithm for predicting consequences ....................... 86
   6.4. A model of a military example .................................. 98

7. HUMAN-FACTOR CONCEPTS AND SYSTEMS ARCHITECTURE .......... 103
   7.1. Human-computer interaction ................................. 105
   7.2. Concepts .......................................................... 107
   7.3. Functional DDSS design ....................................... 110
   7.4. System architecture ............................................. 113

8. EXAMPLE OF A DDSS .................................................. 119
   8.1. Preconditions .................................................. 119
   8.2. Software design and functionality .......................... 121
   8.3. The user interface .............................................. 127
   8.4. Purpose of DECSUP .............................................. 136
   8.5. Data integrity and security ................................. 141

9. EVALUATION ......................................................... 147
   9.1. Introduction ................................................... 147
   9.2. Alternative approaches to evaluation ....................... 148
   9.3. Summary .......................................................... 160

10. DISCUSSION ......................................................... 163
    10.1. Review .......................................................... 164
    10.2. Conclusions and prospect .................................... 168
War is a catastrophic failure. In an area of declarations of withdrawals and arms reduction it is easy to forget that. Change is a great destabilizer, and the need to continue to deter remains. Resources, decisions and readiness hold the key. With all these three under pressure it becomes even more vital to demonstrate commitment to the fighting soldier by providing him with the equipment and support to give him the best possible “fighting chance” on the day his skills and courage will be called upon.

The Right Honorable The Lord Carrington, former Secretary General of NATO.

1. INTRODUCTION

Modern Command and Control systems and foreign affairs operations represent special cases of a more general phenomenon: having the information and authority for decision-making distributed over several individuals or groups. Distributed decision-making can be found in such diverse settings as voluntary organizations, multinational corporations, diplomatic corps, government agencies, and married couples managing a household.

This thesis addresses the problem of structuring a set of decision-makers and communication links in a manner which leads to effective distributed decision-making in the context of military Command and Control systems.

To study the decision-making processes in military headquarters, questionnaires were issued during the WINTER-CIMEX 89 exercise, embodying knowledge and experience of experts from most NATO nations (Schlickmann [2]). The study is based on this inquiry.

Decision makers, at the strategic, tactical or operational level, generally face problems which require more than simple information retrieval. There is usually an associated degree of uncertainty. This makes decision-making difficult. Though we cannot predict the future with certainty, it is
possible, using appropriate methodologies, to identify potential ranges of outcomes and calculate relative likelihoods that particular outcomes will occur.

To reduce uncertainty in connection with decision-making in a decentralized environment, extensive efforts are made to bring all the computing assets of a large military headquarters into a unified framework. This thesis derives from part of these efforts. Readily accessible databases and networks allow efficient access to ranges of information for each commander involved in the decision-making process.

1.1. Defining the issue

The distribution of information, authority, personnel, and resources is part of the reality facing military units, companies with dispersed sales forces, forest-fire fighters, diplomatic services, negotiating teams and many other organizations. In addition to daunting logistics and other technical problems, these organizations face fundamental difficulties in making and coordinating decisions that will serve the interests of both the organization as a whole and the individuals within it. Part of this problem is the tension between the need to control individuals involved and the need to let them respond to the demands of their own immediate situations; another part is the difficulty of translating overall objectives into terms that will be meaningful in the diverse concrete situations encountered throughout the organizations; and another part is the challenge of creating incentive systems that will properly motivate personnel.

As the power of data-processing equipment increases, more attention is being paid to possibilities of managing such large, complex decision-making structures automatically. Military headquarters embody such structures. One technological approach supportive of decision-making in military structures might be the use of multiple processors working together. The practicability of such an approach stems from the fact that each military commander is concerned with specific tasks. Communication with other commanders is needed for coordination purposes.
However, progress in the development of practical means to design a
decentralized decision support system has been lacking. Such systems
(Carley [3], Watson and Buede [4], and Lanir [5]) have so far usually been
developed using methodologies that have evolved in relation to the
design of centralized decision-making systems. There are obviously
fundamental differences between centralized and decentralized
decision-making scenarios. These differences therefore preclude
straightforward transfer of concepts from the centralized to the
decentralized case.

Broadly speaking, the primary motivation of this study of distributed\footnote{1} decision-making systems is the desire to find ways of managing an
organization significantly more complex than one decision-maker can
handle unaided.

1.2. Decision support and the military

Decision-making is a complex but critical management process in a
multidimensional organization like a military headquarters. It necessitates
determination of the objectives and operational requirements at various
levels of command.

The military has preconceptions as to how the decision-making
procedures in relation to well- and ill-structured decision problems (for
definitions see Bots [6], section 2.2) in large military headquarters should be
supported. Their idea is that alternatives at each level of command can be
assessed with reference to their shortcomings and advantages, and defined
in an equilibrium between them.

An example which explains the background of this preconception is as
follows: A group of advisers is told to propose a plan in response to an
international crisis. Intelligence sources reveal that country X is
responsible for a terrorist bombing in country Y, and a group of
government advisers is asked to propose a response to the attack. Once

\footnote{1}{The term ‘distributed’ is used to indicate coordinated, cooperative
decision-making, and includes fully decentralized situations where
decision-makers act independently.
this proposed plan is formulated, other decision bodies, of a formal or ad hoc nature, are given the opportunity to approve or disapprove it. The Joint Chiefs of Staff, the Central Intelligence Agency, the State and Defence departments, and other groups may all have formal or informal involvement in approving or disapproving the plan, with the president or parliament having final authority. The final decision to accept or reject the plan is some function of the advice received.

Whatever sequence of decisions leads to acceptance or rejection of the plan, the decision confronting the group of advisers is: What plan represents the wishes of the group best? Different advisers may have different opinions about how to respond to the attack. Some may wish to do nothing, others may wish to use military forces of some kind. If differences of opinion exist, they are reconciled by using a voting procedure, for example, simple majority rule, weighted majority rule or something similar.

This is the military situation. The military preconception of its problem is, however, different in that it foresees a balance between advantages and disadvantages in relation to any choice. The group making a proposal for acceptance or rejection to a decision-making body can make such a choice. But the decision-making body cannot in general judge the proposal made to it on the same basis. Therefore the requirement for an equilibrium between shortcomings and advantages cannot be met.

1.3. Command and Control

A military Command and Control (C2) system is a decision-making network that reflects a hierarchical organization of C2 cells\(^2\). It is an integrated system comprised of doctrine, procedures, organisational structure, personnel, equipment, facilities and communications that provides authorities at all levels with timely and adequate data to plan, direct and control their activities.

---

2 In this study, 'cell' is taken to mean any organizational unit with a defined role in military Headquarters, regardless of its size or position in the organizational hierarchy.
Chapter 1

Each C2 cell is responsible for the management of some portion of the available resources, where the higher level cells are responsible for a correspondingly greater portion of the resources. Within a C2-system, both command and control decision-making occurs at every level of the hierarchy. Strategic decisions at one level determine how to satisfy the tactical decisions at a lower level. For instance an Army Corps commander may determine that Division X will engage in a frontal assault along corridor Y. This will result in various decisions regarding the positioning and timing of the attack, along with the allocation of various resources to support the attack.

While C2 decision-making occurs at every level of a C2 hierarchy, one factor that does not distinguish between the levels is time pressure. As one moves from higher to lower levels of decision-making, both the time available for decision-making and the time horizon over which decision-making applies decreases. In a conventional conflict, for instance, senior commanders will be primarily concerned with the determination of campaign plans with time horizons that may range from days to weeks. During the execution of a campaign, periodic orders (daily) will be transmitted to the next lowest echelon, which in turn will periodically (every two hours) send new orders to its subordinate echelons, etc., until we get to the forces that actually execute the orders. These decision-makers often deal with the real-time decisions under the severe time pressure of the ongoing battle.

Decision-making under time pressure in Command and Control is a major problem. It always involves a risk of incorrect actions. For example, in case of execution of a SIOP (Single Integrated Operational Plan) in response to a nuclear attack, there would be a tendency to act promptly while Command and Control systems are still intact; that is, a preplanned action must be decided upon early without possibility of later modification. The normal solution in relation to such a situation is to make the systems endure, so that crucial decisions are postponed for as long as possible. An alternative may be the substitution of preplanned execution of decisions by more flexible actions. For example, a retaliatory response might be automatically executed in accordance with predefined strategic guidelines and incoming data on threat and damage assessment.
Chapter 1

Whether such an alternative can be implemented, or is desirable, is still a matter of debate (Zraket [7]).

Another problem caused by the complexity of modern nuclear and conventional forces, on strategic as well as on tactical levels, is that inherent vulnerabilities of C2-systems are almost unavoidable and of great concern to the military.

Possible threats to C2-systems include collateral nuclear weapon effects, such as electromagnetic pulse, and attack on satellite assets, which could disrupt both launch control centres and global military communications. Command and Control modernization and upgrading therefore involve a great amount of 'radiation hardening' of existing facilities and deployment of new, redundant communication systems. Nevertheless, the C2 systems depend crucially on high technology, with computers and other electronics as essential components while problems of vulnerability are likely to persist.

1.4. The Key Tasks

Decision-making analysis in a military environment generally involves working from a number of natural-language requirement documents with limited opportunities for interaction with the decision-makers. These documents may include a list of all decisions which must be made in each stage of a military action as well as the requirements for those tasks which are to support the decision maker by providing pertinent information.

In principle it is not difficult to analyze a commander's mission, and from it derive a set of the necessary C2 decisions and staff actions. In practice, it is problematic, because there is no self-evident set of terms to describe possible decisions and actions, nor is there a set which is indisputably the best.

It is therefore helpful to have available a set of agreed standard actions so that an analyst has a framework in which to operate. These must be carefully chosen; they must describe types of command decisions and staff actions which are essential for mission accomplishment; they must be
designed to allow analysis down to the level of detail needed in practice in many different headquarters. Because of their essential role in C2, these decisions and actions have been given the name ‘Key Tasks’.

From the mission, roles, and responsibilities of a commander, a set of decisions and staff actions can be derived which make up the Command and Control operational requirements and the Key Tasks which follow from it. The C2 operational requirements determine the C2 decision-network (as defined in 13) requirements and these in turn determine the requirements for the components of that network, i.e. the C2 information systems, organization, procedures, facilities, equipment and personnel. This relationship is shown in Figure 1-1.

![Diagram](image)

Figure 1-1: Derivation of C2-systems requirements
Key Tasks are chosen to express types of necessary decisions and staff actions in terms of results; they are not concerned with the manner or means by which they are carried out. Thus a Key Task will be applicable to many headquarters, even though each headquarters may complete the task differently in terms of detailed procedures. Analysis based on Key Tasks chosen in this way provides comparability between headquarters, which helps determine how to provide effective Command and Control.

Key Tasks do not describe how a decision is made, nor how staff actions are completed, and are therefore insufficient in themselves to determine quantitative features of C2 operational requirements. An essential feature of the Key Task concept is that each Key Task is associated with one particular phase of the C2 Cycle (Athans [8]), (see Figure 1-2).

The first three phases of the cycle can be thought of together as a supporting phase.

**Maintenance** of status covers the provision, gathering, storage, consolidation and validation of information for transmission in whole or in part, processed or unprocessed, to other parties.

The next phase, **Assessment**, comprises the evaluation, comparison, analysis and review of the data provided in the first phase on the basis of directions and tasks obtained by the commander. Such tasks deal with the overall command mission, and with all factors concerned with the situation, capabilities, and courses of action, both actual and foreseen, of friendly and enemy forces. Special circumstances may demand that the results of some tasks in this phase be fed directly to the Decision or Execution/Coordination phases.

**Planning**-phase work is based on the results of the assessment phase, and results in alternative plans for various courses of action. Prior to the commander's decision, many planning tasks will be completed in as much detail as possible. Decision support for this phase should be designed to act interactively, to allow the creativity of staff officers to combine with information processing capabilities to take all relevant figures into consideration, and to calculate the best options.
Figure 1-2: Command and Control Cycle

The **Decision-making** phase demands the direct participation of the commander. Decisions implement significant plans or directions, direct mission changes of subordinates, or make significant requests to higher headquarters.
Chapter 1

The **Execution** phase will be to some extent a matter of judgment, depending on the command's concept of operations and the extent to which authority to make important decisions is delegated.

The transformation of a headquarters mission into a set of Key Tasks, or a decision as to whether or not a necessary headquarters activity is a Key Task, is not automatic, but inevitably to some extent a matter of judgment.

For a decision or action to qualify as a Key Task it must meet the following criteria:

1. It must be an activity which is essential to the performance of a commander's mission during tension or war, or one which directly contributes to the fighting ability of a command.

2. It must relate to
   a) a decision, normally taken by the commander or a senior subordinate, concerning the implementation of important plans or directives, changing the intensity or scope of the conflict, directing mission changes in the mission of subordinates, or conveying important requests to higher headquarters,
   or
   b) a staff action such as arriving at a routine decision on the basis of a commander's decision or guidance, issuing directives, developing, reviewing, refining and coordinating assessments and plans, preparing and presenting recommendations, or obtaining and maintaining required information.

3. It must describe the purpose of the activity, not the means or method of its execution. That is, it must be independent of any specific organization or headquarters. It must normally be confined to one phase of the C2 cycle (Figure 1-2).
1.5. Outline of study

Decision-making models have been studied from the economic, management theoretic, and military point of view. What is urgently needed is an approach to model distributed decision-making in actual, complex work settings (Defense Research Group [9, 10]). Such models should present the information and communication needs together with the information processing models and strategies of different decision makers cooperating in the control of a loosely coupled work domain.

Analysis of decision procedures which are not centrally controlled can be done by observation at two levels explained below. Analysis is facilitated by the use of procedures established in the course of the Command and Control cycle. The C2 cycle is employed at the organizational level for coordination of decisions and at the cell level for reaching decisions. At the organizational level, coordination of individual decisions is investigated. At the cell level, the decision-making process in the individual cells and communication between the cells are considered.

It is shown that control of an organization with distributed decision-making processes can be achieved by specific coordination strategies. These strategies are developed in this study, and their utilization in relation to various forms of military organizations is investigated. For each cell of a military headquarters a uniform decision-aiding algorithm is presented, which takes account of the constantly changing frequency of communication with neighbouring cells.

Chapter 2 describes the definitions and techniques used to analyze the decision domain and to develop a prototype of a distributed decision support system (DDSS). In Chapter 3 the decision-making process is reviewed and the relationships between decision-making and information systems are discussed. Characteristics of and constraints relating to human information processing are listed in connection with the identification of human constraints in relation to decision-making.

Chapter 4 describes a prototype methodology for development of a DDSS. It discusses some approaches to modelling a DDSS and suggests design
principles. The model and tools chosen for implementation of a DDSS are recorded.

The subsequent chapters deal with requirements analysis, system design and architecture, definition of a model incorporating decision support algorithms, and finally implementation and evaluation of a DDSS in the specific case of a military headquarters.

Requirements analysis of decision-making processes with examples in relation to Key Tasks is described in Chapter 5. The outcome is a framework of cells on which a DDSS can be built.

Chapter 6 discusses strategies for distributed decision-making in relation to certain kinds of organizational structures. An algorithm for predicting the consequences of decisions taken in the individual cells of such structures is presented and its application illustrated in relation to a Key Task in a crisis-management situation.

The architecture of a DDSS, presented in Chapter 7, takes account of human decision-making limitations. A basic DDSS configuration is suggested and a DDSS architecture for an environment involving distributed decision-making is proposed.

Chapter 8 describes a prototype of a DDSS. Integrity and security aspects are considered. In Chapter 9 the prototype system is evaluated.

Chapter 10 summarizes and assesses results in relation to the need, stated in 1.1. It makes suggestions for further research.
Chapter 2

2. DEFINITIONS AND TECHNIQUES

This Chapter discusses techniques on which the hypothesis this study seeks to test is based, and discusses their properties. Some definitions that should facilitate the understanding of later on described model components are given. Programming methods are described in detail and some definitions relating to the syntax of the programming language used are developed. Logic programming is introduced and the rationale for logic modelling is presented as a consequence of development in logic programming.

2.1. Definitions from lattice theory

This section introduces the requisite concepts and results concerning monotonic mappings and their fixpoints in relation to lattices (Lloyd [11]).

**Definition** Let $S$ be a set. A relation $R$ on $S$ is a subset of $S \times S$.

We usually use infix notation writing $(x,y) \in R$ as $xRy$.

**Definition** A relation $R$ on a set $S$ is a partial order if the following conditions are satisfied:

(a) $xRx$, for all $x \in S$.
(b) $xRy$ and $yRx$ imply $x=y$, for all $x,y \in S$.
(c) $xRy$ and $yRz$ imply $xRz$, for all $x,y,z \in S$.

We adopt the standard notation and use $\leq$ to denote a partial order. Thus we have (a) $x \leq x$, (b) $x \leq y$ and $y \leq x$ imply $x=y$ and (c) $x \leq y$ and $y \leq z$ imply $x \leq z$, for all $x,y,z \in S$.

**Definition** Let $S$ be a set with a partial order $\leq$. Then $a \in S$ is an upper bound of a subset $X$ of $S$ if $x \leq a$, for all $x \in X$. Similarly, $b \in S$ is a lower bound of $X$ if $b \leq x$, for all $x \in X$. 
**Definition** Let \( S \) be a set with a partial order \( \leq \). Then \( a \in S \) is the least upper bound of a subset \( X \) of \( S \) if \( a \) is an upper bound of \( X \) and, for all upper bounds \( a' \) of \( X \), we have \( a \leq a' \). Similarly, \( b \in S \) is the greatest lower bound of a subset \( X \) of \( S \) if \( b \) is a lower bound of \( X \) and, for all lower bounds \( b' \) of \( X \), we have \( b' \leq b \).

The least upper bound of \( X \) is unique, if it exists, and is denoted by \( \text{lub}(X) \). Similarly, the greatest lower bound of \( X \) is unique, if it exists, and is denoted by \( \text{glb}(X) \).

**Definition** A partially ordered set \( L \) is a complete lattice if \( \text{lub}(X) \) and \( \text{glb}(X) \) exist for every subset \( X \) of \( L \).

**Definition** Let \( L \) be a complete lattice and \( T : L \to L \) be a mapping. We say \( T \) is monotonic if \( T(x) \leq T(y) \), whenever \( x \leq y \).

**Definition** Let \( L \) be a complete lattice and \( X \subseteq L \). We say \( X \) is directed if every finite subset of \( X \) has an upper bound in \( X \).

**Definition** Let \( L \) be a complete lattice and \( T : L \to L \) be a mapping. We say \( T \) is continuous if \( T(\text{lub}(X)) = \text{lub}(T(X)) \), for every directed subset \( X \) of \( L \).

By taking \( X = \{x, y\} \), we see that every continuous mapping is monotonic. However, the converse is not true. Next we study fixpoints of mappings defined on lattices.

**Definition** Let \( L \) be a complete lattice and \( T : L \to L \) be a mapping. We say \( a \in L \) is the least fixpoint of \( T \) if \( a \) is a fixpoint (that is, \( T(a) = a \)) and for all fixpoints \( b \) of \( T \), we have \( a \leq b \). Similarly, we define greatest fixpoint.

The next result is a weak form of a theorem due to Tarski [12].

**Proposition 2.1** Let \( L \) be a complete lattice and \( T : L \to L \) be monotonic. Then \( T \) has a least fixpoint, \( \text{lfp}(T) \), and a greatest fixpoint, \( \text{gfp}(T) \). Furthermore, \( \text{lfp}(T) = \text{glb}\{x : T(x) = x\} = \text{glb}\{x : T(x) \leq x\} \) and \( \text{gfp}(T) = \text{lub}\{x : T(x) = x\} = \text{lub}\{x : x \leq T(x)\} \).
Chapter 2

**Proof** Put $G=\{x: T(x)\leq x\}$ and $g=\text{glb}(G)$. We show that $g \in G$. Now $g \leq x$ for all $x \in G$, so that by the monotonicity of $T$, we have $T(g) \leq T(x)$, for all $x \in G$. Thus $T(g) \leq x$, for all $x \in G$, and so $T(g) \leq g$, by the definition of glb. Hence $g \in G$.

Next we show that $g$ is a fixpoint of $T$. It remains to show that $g \leq T(g)$. Now $T(g) \leq g$ implies $T(T(g)) \leq T(g)$ implies $T(g) \in G$. Hence $g \leq T(g)$, so that $g$ is a fixpoint of $T$.

Now put $g' = \text{glb}\{x: T(x)=x\}$. Since $g$ is a fixpoint, we have $g' \leq g$. On the other hand, $\{x: T(x)=x\} \leq \{x: T(x)\leq x\}$ and so $g \leq g'$. Thus we have $g=g'$ and the proof is complete for lfp($T$).

The proof for $\text{gfp}(T)$ is similar. Q.E.D.

**Proposition 2.2** Let $L$ be a complete lattice and $T: L \rightarrow L$ be monotonic. Suppose $a \in L$ and $a \leq T(a)$. Then there exists a fixpoint $a'$ of $T$ such that $a \leq a'$. Similarly, if $b \in L$ and $T(b) \leq b$, then there exists a fixpoint $b'$ of $T$ such that $b' \leq b$.

**Proof** By proposition 21, it suffices to put $a'=\text{gfp}(T)$ and $b'=\text{lfp}(T)$. Q.E.D.

The definitions on lattice theory formulated in this section serve as a basis for the definitions of finite automata and organizational structures.

**2.2 Logic programming**

This section provides a brief introduction to logic programming, followed by a discussion on syntax as well as declarative and procedural semantics of logic programs.

The credit for the introduction of logic programming goes mainly to (Kowalski [13]) and (Colmernauer [14]). In 1972, Kowalski and Colmernauer were led to the fundamental idea that logic can be used as a programming language. The acronym PROLOG (PROgramming in LOGic) was conceived, and the first PROLOG interpreter [14] was implemented.
The idea that first-order logic, or at least substantial subsets of it, could be used as a programming language was revolutionary because, until 1972 logic had only ever been used as a specification or declarative language in computer science. However, what [13] shows is that logic has a procedural interpretation, which makes it very effective as a programming language.

One of the main concepts of logic programming, which is due to Kowalski ([15] and [16]), is that an algorithm consists of two disjoint components, the logic and the control. The logic is the statement of what the problem is that has to be solved. The control is the statement of how it is to be solved. Generally speaking, a logic programming system should provide ways for the programmer to specify each of these components. However, separating these two components brings a number of benefits, not least of which is the possibility of the programmer having only to specify the logic component of an algorithm and leaving the control to be exercised solely by the logic programming system itself. In other words, an ideal of logic programming is purely declarative programming.

2.2.1. Syntax of logic programs

We use the Edinburgh syntax (Bowen [17]) for logical variables, terms, and predicates:

**Definition** A term is a variable (e.g. X) or a function symbol of arity \( n \geq 0 \), applied to \( n \) terms (e.g., \( c \) and \( f(a, X, g(b, Y)) \)).

**Definition** An atom is a formula of the form \( p(T_1, \ldots, T_n) \), where \( p \) is a predicate of arity \( n \) and \( T_1, \ldots, T_n \) are terms.

**Definition** A definite clause (clause for short) is a formula of the form

\[
A \leftarrow B_1, \ldots, B_n, \quad n \geq 0,
\]

where \( A \) is an atom, and \( B_1, \ldots, B_n \) is a sequence of atoms. \( A \) is called the clause's head; and \( B_1, \ldots, B_n \), its body. We denote the empty sequence of atoms by true.
**Definition** A logic program is a finite set of definite clauses.

**Definition** A goal is a sequence of atoms $A_1, A_2, \ldots, A_n$. A goal is empty if $n = 0$, atomic if $n = 1$, and conjunctive if $n > 1$. Each atom in a goal is called a goal atom. A goal atom is often called a goal for short.

**Definition** The vocabulary of a logic program $P$ is the set of predicates and function symbols that occur in the clauses of $P$.

### 2.2.2. Informal semantics of logic programs

Logic programs can be read both declaratively and procedurally. We describe these two views here informally.

Declaratively, each clause in a logic program is read as a universally quantified implication. If $X_1, X_2, \ldots, X_n$ are the variables in the clause $A \leftarrow B_1, \ldots, B_k$, then the clause is read 'for all $X_1, X_2, \ldots, X_n$, $A$ is true if $B_1$ and $B_2$ and ... and $B_k$ are true'. A logic program is read as the conjunction of the universal implications corresponding to its clauses.

Procedurally, logic programs can be viewed as an abstract computational model, like the Turing Machine, the Lambda Calculus, and the Random Access Machine. A computation in this model is a goal-driven deduction from the clauses of the program. Like the non-deterministic Turing machine, computations in this model are non-deterministic: From each state of the computation, there may be several possible transitions. Specifically, the clauses of a logic program can be read as transition rules of a non-deterministic transition system.

The state of a computation consists of a goal (sequence of atoms) $G$ and a substitution (assignment of values to variables) $F$, and is denoted by a pair $(G,F)$. A computation begins with an initial state consisting of the initial goal to be proven and progresses from state to state according to the following transition rules, 'Reduce' and 'Fail'.

A computation can be viewed as an attempt to prove the initial goal from the program. At each state the goal represents a statement whose proof
will establish the initial goal. The substitution represents the values computed so far for variables used in the computation, including the initial goal variables.

A computation ends in a state whose goal is either true or fail. In the former case, the computation is successful and corresponds to a successful proof of the initial goal. In the latter, it fails. The substitution in the terminal state, restricted to the variables in the initial goal, is called the answer substitution of the computation.

A successful computation has the property that its initial goal, instantiated by the answer substitution, is a logical consequence of the program.

The 'Reduce' and 'Fail' transition rules require that the variables in the clause be consistently replaced by new variables that have not been used before in the computation.

A computation progresses until it reaches a terminal state, which is a state to which no transition applies. By the definition of 'Reduce' and 'Fail', the goal in a terminal state is either true or fail.

2.2.3. Declarative versus procedural style of programming

We have shown the dual reading of clauses: declarative and procedural. How do they interrelate when composing logic programs? Pragmatically, one thinks procedurally when programming. However, one thinks declaratively when considering issues of truth and meaning.

An example of a highly procedural representation is a data processing program. The representation of the problem, and the procedure for solving it, are closely intertwined. A spreadsheet package with a goal-seeking feature is an example of a more declarative representation. It declares the structure of the problem, but not the sequence of steps for solving it. The solution procedure is built in.

We should like to give another example of declarative and procedural reading in PROLOG. A meta-interpreter for a language is an interpreter for
Chapter 2

the language written in the language itself. The ability to write a meta-interpreter easily is a very powerful feature for a programming language. It enables the building of an integrated programming environment and gives access to the computational process of the language. Since a meta-interpreter is a PROLOG program, we give a relation scheme. The relation solve(Goal) is true if Goal is true with respect to the program being interpreted.

The following meta-interpreter simulates the computational model of logic programs. Goal reduction for PROLOG programs can be described by the three following clauses.

Program 2-1.

\[
\text{solve}(\text{Goal}) \leftarrow \\
\text{Goal is deducible from the PROLOG program defined by} \\
\text{clause/2.} \\
\text{solve(true)}. \\
\text{solve}(\text{(A,B)}) \leftarrow \text{solve(A)}, \text{solve(B)}. \\
\text{solve(A)} \leftarrow \text{clause(A,B)}, \text{solve(B)}. \\
\]

Program 2-1: A meta-interpreter for PROLOG

Declaratively, the interpreter reads as follows: The constant true is true. The conjunction (A,B) is true if A is true and B is true. A goal A is true if there is a clause A ← B in the interpreted program such that B is true.

We give also a procedural reading of the three clauses in Program 2-1. The solve fact states that the empty goal, represented in PROLOG by the atom true, is solved. The next clause concerns conjunctive goals. It reads: 'To solve a conjunction (A,B), solve A and solve B'. The general case of goal reduction is covered by the final clause. To solve a goal, choose a clause from the program whose head unifies with the goal, and recursively solve the body of the clause.

Along this dimension from declarative to procedural, logic is a prime example of a declarative representation. Unlike non-procedural application
packages and some spreadsheets, which are declarative but limited to a certain class of problems, logic and logic programming solutions are broader in scope.

One of the principal advantages of declarative representation is its modularity. The argument for modularity in software design is that it yields the ability to develop or modify one module without having to consider the impact on other parts of the program. The effects are limited to the arguments in the subroutine call. The advantages of structured programming are also of this type. By forcing the control structure of the program to a hierarchical sequence of calls, one limits the possibility of unforeseen side-effects in making a modification.

Logic programming may be regarded as an extension of this trend - taking modularity down to the level of individual statements (predicate definitions). As an extreme form of modularity, the possibility of unexpected side-effects occurring is greatly reduced.

Logic programming, like structured programming, enforces a strict discipline on programming style, often thereby requiring greater modelling skill than with conventional languages. In contrast to conventional languages, the notation of logic and thus logic programming is one especially developed over centuries for its expressiveness and perspicuity. The role of logic in philosophy, developed as a tool for clarifying arguments, is to make clear the essential aspects of a debate, and to reduce deduction to a finite number of verifiable, mechanical operations.

The other major objection to declarative representations generally, and logic programming in particular, is with regard to computational efficiency. Because the solution procedure in logic programming is domain independent, it is not capable of exploiting a special problem structure. For deterministic problems, it may waste a great deal of time exploring irrelevant alternatives.
2.3. Logic modelling and decision-making

Although a variety of decision-making models have been studied, mathematical programming models and stochastic models have been perhaps the most favoured (Buckley [18]). Recent developments in information-systems technology and formal logic have provided a basis, on which logic is becoming a valuable tool for modelling problems of interest to developers of information systems. Consequently, a new field of research, called 'logic modelling', has begun to emerge, e.g. see Kimbrough and Lee [19], Widmeyer [20], and Nute [21]. In this study, we follow this rapidly developing area of research.

2.3.1. Why logic?

Today, logic is interesting not merely because it studies truth-preserving inferences. The main reason is that studies of logic (Gabbay and Guenther [22]) have produced a series of formal languages (called 'logics', as in propositional logic, predicate logic, modal logic, fuzzy logic and so forth). These formal languages have a number of properties that are of interest to modellers. First, many of these languages are well understood from a logical point of view, so that it is often possible to determine by a mechanical routine whether an inference expressed in these languages is indeed valid.

For example in predicate calculus, given any model expressible in predicate logic and given any conclusion which validly follows from the model, there is a general, finite, mechanical routine for showing that the conclusion actually does follow (Kimbrough [19]). This remarkable fact led to the concept of logic programming. The idea is that a formal language for a logic could also be used as a programming language (Kowalski [15]).

2.3.2. Limitations of established modelling techniques

Modelling techniques for designing decision support systems, such as mathematical programming and stochastic approaches, have been criticized in Brennan [23] and Sprague [24] because:
Chapter 2

- Their output is not easily understood by the average model user.

- They cannot cope with discontinuities, especially in model input data.

- Model validity cannot be assessed on the basis of model output.

- They are unable to guide the user through possible analysis beyond the output of the models.

- Models have not included all necessary variables.

- Interaction between the model and the user is minimal.

- All of the complex advantages and disadvantages of the various alternatives must be summarized in the form of a utility function on real values.

Some of the difficulties encountered in applying these modelling techniques have stimulated a growing research interest in models using vector-valued, multi-criterion, and other objective functions - e.g. Barron and Schmidt [25], Lootsma [26], Farquhar and Nakamura [27], and the extensive bibliography in Buckley [18].

2.3.3. Application aspects

Mathematical programming and stochastic models rely mainly on numeric representations and arithmetic forms of reasoning. They tend to emphasize problems whose principal attributes are scalable, e.g., as on a ratio, interval or ordinal scale. The reasoning lies largely on the magnitude of a given set of attributes.

Logic, by contrast, supports natural reasoning on the qualitative aspects of a problem. It supports reasoning about features that cannot easily be modeled numerically. Logic may also be used for quantitative modelling. But in these cases, highly efficient arithmetic reasoning procedures are available. Such numeric methods, however, exist only for certain kinds of
decision-problems, i.e., for those in which a scaling of total ordering on attribute values is available (Roberts [28]). For decision-problems showing a weaker class of orders, e.g. various forms of partial orders, non-numeric methods such as logic programming must be used. The paper by Widmeyer and Lee [20] discusses these problems with respect to preference orderings. More broadly, proper modelling of qualitative reasoning is regarded in much of the artificial-intelligence literature as essential for further progress in information systems (Rich [29]).

2.3.4. Logic modelling: Other applications

Example 1:
Negotiation is an important means of reaching a decision. Negotiation has traditionally been modelled quantitatively. Matwin [30] has described a new approach allowing a negotiation problem to be represented by a logic program. The rationale for and the consequences of this new approach to representation of a negotiation problem are discussed below. A negotiation support system, NEGPLAN, has been prototyped in PROLOG and continues to be used because it reflects the association between static and dynamic aspects of negotiation better than quantitative methods.

Negotiation is a form of decision-making based on information exchange. Communication between the negotiating parties has a decisive influence on the outcome of negotiations. One approach to negotiation modelling and system support is to express it as an optimization problem and apply to it standard methods of mathematical programming. The use of standard methods may be cumbersome, since these models typically involve some built-in mechanism of choice between different decision alternatives. Any such approach may force the user to specify his decision problem in greater detail than he would like to or even be able to consider.

The approach outlined in Matwin et al. [30] is based on logic, which allows modelling of various stages of negotiations in a flexible way. The purpose of their system was to help the negotiator to prepare the negotiating position and allow him to experiment with different assumptions. They provided a logic-based model (logic program) and an interpretation
mechanism. Their view of the negotiation process was as a goal-oriented activity whose ultimate objective is to 'win' the negotiations. Although the precise sense of 'winning' was not defined, it was intuitively understood to mean achieving the highest-level goal.

Example 2:
Space logistics is evolving. As experience of launch and recovery of space systems accumulates, maintenance and refurbishment activities have increased. Effective development of a permanently staffed space station requires consideration of logistic support activities during the design phase of the station. The Space Station Operations Model (OPSMODEL) is a decision-support system developed using a logic modelling approach by the NASA Langley Research centre and Computer Sciences Corporation (Luxhoj [31]).

OPSMODEL has two components: a database manager and an operations simulator. The database manager uses R:base 5000. The operations simulator uses AAIS PROLOG. Discrete event simulation is used. A model database is created using R:base 5000. It is then transferred to the simulator. Simulation results are output to the database. The model database is then updated. R:base 5000 manages OPSMODEL output summaries.

OPSMODEL is a DSS because it is interactive, because different scenarios can be evaluated, and because the users can undertake single or multi-parameter sensitivity analysis (facilitating 'what if' analysis). The effects of crew-skill profiles on time-to-complete variation could be analyzed, for example. Decision makers could be concerned with the effects on power consumption of alternative mission sets in relation to payloads. Crew-sizing problems could be studied. The effects of alternative space station layouts on travel times could be analyzed. An examination of the power profile after a simulation might suggest a need for additional solar collectors on the space station.

The OPSMODEL DSS is used to develop the integrated logistic support (ILS) program for the space station as outlined in Hosmer [32]. Simulation is desirable since in-orbit time will be at a premium and must be efficiently
scheduled. Operational costs must be minimized and productivity of the station maximized.

To summarize: Developments in logic and logic programming have converged to the point at which logic is, for a variety of problems, a useful modelling tool. However, we feel that in none of the applications areas that we have touched it can be said that the logic modeling approach is mature. Much remains to be learned, but we believe the basic approach: Modelling with logic will likely offer exciting opportunities.
3. EXPLORATION OF DECISION-MAKING

This Chapter starts by describing the decision-making process before exploring the relationships between decision-making and information systems. The Chapter identifies human limitations in relation to decision-making and concludes the text by providing some projections for the future.

3.1. Decision-making steps in relation to information systems

Simon [33] defines the three stages in the decision-making process as intelligence, design, and choice. A fourth stage (see Bots [34] and Sol [35] for further discussions), implementation, may be added to these three stages. Figure 3-1 illustrates these four stages.

![Diagram of decision-making process]

Figure 3-1: Four stages in the decision-making process

**Intelligence**
In the intelligence phase, the environment of an organization is studied for conditions requiring decisions. Data are collected from a variety of sources (internal and external) and are processed. From this information, the decision-maker may discover ways of approaching his or her problem.

**Design**
In the design stage, the objective is to generate alternatives, evaluate different alternatives creating courses of action, and evaluate the
feasibility and accessibility of each solution.

Choice
The choice phase is usually straightforward. Results of alternatives are simulated and the best and most effective course of action is chosen and then implemented. For example, if an air threat is present but its location is not known, we may choose the first given alternative, which might be put 2 early warning aircraft into the air and separate the aircraft by about 120 degrees.

Bots [34] explains implementation as ‘implement solution’. In other words putting the chosen solution into practice. It may be followed by a next step such as evaluate selected solution in light of outcome. It may also be compared with the execution phase from the command and control cycle shown in Figure 1-2.

3.2. The role of information systems in decision-making

Information has one main use: To provide a basis on which to make decisions. Information is the data, opinions, forecasts and facts used in decision-making. Information systems utilize these information resources for people or computer systems to make decisions. They cover not only the capture, storage and communication of information but also the application of knowledge and skills to decision-making.

Decision-making is fundamental to the operation and survival of any organization since it provides the means by which organizations are able to respond to the changing political, social, business or military environment. Every action or initiative arises out of a decision. Even doing nothing is a decision to not do anything.

Information systems are important because they are concerned with organizing the raw materials for decision-making: Information. Many terms have been used to describe basically similar information systems, and many dissimilar information systems have been described using similar terms, (see Sol 36). To avoid terminological confusion and allow relevant design issues to be clearly and unambiguously discussed, a set of
commonly accepted terms is needed.

In this study, distributed decision-making (DDM) is defined as a process in which each cell\(^3\) uses its perceptions formed on the basis of limited information about a global state to arrive at decisions intended to result in the achievement of global objectives.

An important point in relation to this definition is that information systems which support distributed decision-making are interactive, and augment decision-maker's abilities to make decisions. A distributed decision support system (DDSS) does not make decisions for its user, but may offer suggestions and recommendations. A DDSS must however do more than just make information available to a decision-maker - access to information in itself is not enough.

3.2.1. Information sources used for decision-making

Most information systems are concerned with providing external information to the decision-maker to enable him to use his special knowledge, skills and judgment more effectively in decision-making.

The external information comes to the decision-maker from three main sources, Land [37]. The first, and most direct, is from the real world and consists of observations about what is actually happening. The second is from casual information sources such as colleagues, books, and news reports. The third is from dedicated information systems such as regular management information systems (MIS) and decision support systems (DSS). Dedicated Information systems can be viewed as providing a spectrum of support for decision processes, a concept put forward by Sol [36], (Figure 3-2).

---

3. In this study, 'cell' is taken to mean any organizational unit with a defined role in military Headquarters, regardless of its size or position in the organizational hierarchy.
Figure 3-2: Information sources for decision-making

It is the implementation and use of the third category, dedicated information systems, which is of principal concern to the decision maker, albeit within the context of the first two. It is convenient to view these various systems within the context of the resources and capabilities which humans require to make decisions.

At present, information system development is focussed on collection, communication and storage of facts on the state of affairs in the work domain. Research in the representation and communication of implicit or tacit knowledge of intentions, values and motives is necessary. Early attempts to explore the potential of computers for integrated management information systems failed, probably because designers of those systems mistakenly assumed that executive managers made decisions from factual information in reports and statistics. In fact they spend most of their time
exploring the values and intentions of other executives in direct contact by phone, meetings, and cocktail parties (Mintzberg [38]).

3.2.2. Empirical evidence of information systems for decision support

The information processing systems (MIS, DSS) shown in Figure 3-2 are now used widely to support and, in some cases, to perform decision-making. However, Brennan [39] and Sprague [40] have identified a number of limitations:

- They generally deal effectively only in facts. Information sources such as gossip, ideas, opinions, views, judgments, rules of thumb, experience and predictions are not encompassed.

- They are unable to determine what information is relevant to the decision and hence often overload the user with information.

- They can process data only when algorithms or procedures exist or can be developed which specify how the data is to be processed.

- They hide from the user the approaches, assumptions and reasoning built into the decision models.

- They are difficult to modify and as a result they tend to be unresponsive to changes in user requirements.

- They have difficulty reflecting users' varying cognitive styles.

- They have long development times.

Inroads on these problems, in particular for the last one are beginning to be made through prototyping, application generators, programmers' workbenches and the use of spreadsheet and database packages. However, their impact has not been sufficient to satisfy the needs placed on information processing systems by military users.
3.3. Human factors in decision-making

Recent changes have considerably complicated the task of decision support. Some of these changes have been exogenous, such as increases in the complexity of an organization's environment and the speed of response demanded by it. Neither a fleet threatened by high-speed missiles, nor a multinational corporation facing round-the-clock financial trading has the opportunities for internal consultation and coordination that it might have had in a slower age. Some of these changes are endogenous, reflecting new capabilities for transmitting information to local units and monitoring their behaviour. The pressures and possibilities come from developments such as teleconferencing, electronic mail, satellite communications, shared databases, and the online monitoring of employees at computer terminals. The intention of human-factors research is to anticipate and understand these changes, to shape the design of both the technologies and the organizations that must accommodate them.

Human factors research looks at design and performance issues involving the interface between people and machines. Examples include creating effective workstations, developing communications protocols, selecting and training operators, designing displays to reveal the current status of an industrial facility, and evaluating the human side of system performance. Each problem requires the skills of somewhat different mixes of the professions contributing to human factors, which include psychology, industrial engineering, physical anthropology, applied mathematics, training, and sociology. Each requires collaboration with other specialists' knowledge about the environment in which people perform their tasks.

Distributed decision-making systems require the full range of this expertise. Taking military Command and Control as an example, there is the need to design computer systems components as diverse as communications protocols, symbolic computer displays, filing (and retrieval) schemes for contingency plans, minimally disruptive maintenance schedules, procedures for updating key personnel, specifications for when it is permissible to override computerized controls, and rules for rewarding (or disciplining) personnel. Although the particulars will be very different, roughly the same tasks can be found in
setting up an air traffic control system, an integrated forest fire fighting system, a telephone emergency system, an international commodities trading system, or a carrier-based naval task force.

The differences among these systems clearly call for particularized substantive expertise. However, the similarity of their functions and challenges creates the opportunity to study distributed decision-making systems as a general phenomenon. To meet this challenge, empirical research on human-factor issues in distributed decision systems is needed.

Applying existing theories to distributed decision-making systems requires some perception of which features of their design are most important to their operation. To some extent, that perception may come from existing research or direct experience with particular systems. Clearly, there are many questions requiring direct empirical study to provide direct input to designers. As a first approximation, these topics can be divided into those concerning the behaviour of individuals acting alone, those concerning the interaction of individuals with machines, those concerning the interactions of multiple individuals, and those concerning organizational behaviour - all within distributed decision-making systems (compare Chapter 9 for evaluation results). All have implications for human-factor specialists, helping them either to propose designs or to anticipate the performance of designs proposed by others.

3.4. Characteristics and constraints of human information-processing

A distinct feature of distributed decision-making systems is the interdependence of many parts, each of whose status may be constantly changing. Users within the system must be mindful of the possibility of changes as they decide what actions to take, what information to solicit, and what information to share. Routine reporting requirements are designed to help people keep up to date. However, they may not be appropriate for all circumstances and may create a greater stream of information than can be incorporated in a user's mental picture of the system's status. Standard operating procedures are designed to direct actions in uncertain situations. However, that very uncertainty often leaves some latitude for determining exactly what situation exists at any
moment. It would be very helpful to know how (and how well) people create, maintain, and manipulate their mental pictures of such complex, dynamic systems.

Although computers and other machines are used to make simple decisions (such as deciding when to invoice customers), the vast majority of decision-making is done by humans. However, unaided, humans have a number of limitations as decision makers (Chi, Glaser and Farr [41]):

- They may not be aware that a decision needs to be made.
- They may not be able to acquire all the relevant information to make the decision or even to know what information is relevant.
- They may not know enough about the problem area to be able to determine how the available information impacts on the decision.
- They may not be able to acquire that knowledge because it is not readily available or would take too long to assimilate.
- They may not know how to apply their knowledge so as to be able to reach a decision.

Following the process of decision-making described in the preceding sections, and the work done in [41], an ideal, normatively acting commander should be able to:

- establish a correct mental representation of a decision situation,
- establish a mental model of the problem domain,
- mentally search a problem domain,
- consider new information in an exhaustive and unbiased manner,
- calculate preferences and likelihoods associated with decision outcomes,
- quantify and scale outcome utilities consistently,
- decide rationally, i.e., select an alternative with highest priority.
Chapter 3

Obviously, such demands are not in agreement with human nature. Real commanders have to operate within a framework of constraints of time, knowledge, data, memory, and cognitive resources, see Hink and Woods [42], Sol [36], Wickens and Flach [43], and Nagel [44]. While naive decision-makers at least try to act normatively, experienced decision-makers

- act impulsively,
- use a small set of rules of thumb,
- apply satisfying criteria instead of utility-maximizing criteria,
- neglect statistics and probabilities,
- have difficulty in combining competing attributes or objectives,
- use inconsistent preferences and risk assessment,
- don't consider all consequences of outcome options,
- are overly subject to situational context,
- apply bias and noise to heuristic judgment,
- have problems in analyzing or reasoning,
- have inappropriate confidence in their own decisions,
- use their internal representation whether right or wrong,
- are unable to predict processes correctly.

In summary decision-makers may use global, not analytical assessment. They may make a quick assessment of a decision situation pattern, followed by an immediate categorization of the situation and an associative decision.

Two essential points should be made here. First, it must be emphasized that a particular task can be performed by a particular commander at varying levels of control, depending on his current level of training. Also, a skill achieved in a task by extensive training may be lost when training ceases. A fallback from trained to untrained behaviour can then be observed, associated with longer operating times. Second, it is obvious that tasks requiring very fast reactions can only be mastered on the skill-based level.
3.5. Identifying decision-making constraints

For a DSS to be useful, decision-makers must be able to integrate the computer support given by a DSS into their cognitive process. For example, the success of a DSS depends, not only on the quality and completeness of the knowledge elicited from expert decision-makers. It also depends on the compatibility of the recommendations and decisions given by the DSS with the decision-makers' conceptualization of the problem. In building the DSS, there also may be conceptual conflicts between the developer and the expert decision-maker. Obviously, there are many opportunities for mismatches between the developer's and decision-maker's ideas of the domain, especially with respect to individual and inter-individual differences among users.

Zachary [45] has identified six general decision support needs common to most decision situations. These six general or commonly occurring limits on unsupported decision processes are:

1. Inability to predict processes

People often use visual images to represent problems requiring solutions. They reason about the problem by picturing in their mind's eye the relationships involved. Unfortunately, people are not as good at this projection process as they tend to believe. For example, they may try to represent the trajectories of two aircraft as curves, but cannot accurately project the point of interception without resorting to a piece of paper. The same limitation applies to projections of physical processes in time. A person observing a moving symbol on a screen will not be able to predict its location five seconds later with any accuracy. Real-time decision-making dramatically increases the frequency with which human operators must make such time/space projections. Computers can predict such processes well through numerical simulation. Difficulties in process predictions are therefore often amenable to computer support.
2. Difficulty in combining competing attributes or objectives

In many decision problems the decision-maker has several criteria that can describe an expected outcome of a decision. Because decision-making is a goal-based activity, there will always be some criteria used to determine if and when the goal has been met by the unaided decision process. Often the criteria are ill-defined or clearly suboptimal. In such cases, the decision-maker could benefit from access to an algorithm for evaluation of possible outcomes. Even when the decision maker has clear rules for comparing many possible outcomes to find the decision, the application of these rules can be difficult. A similar problem occurs when the decision-maker is trying to make tradeoffs among competing objectives. Computers can make such computations very efficiently, so difficulties in combining choice criteria are amenable to computer support.

3. Inability to manage information needed in the decision process

Decision makers often fail to make use of all the information available to them simply because they are unable to manage it effectively 'in their head'. A human decision maker can easily be overwhelmed by a large amount of information, and hence fail to process key inputs or fail to recall and apply crucial pieces of knowledge. The ability of computers to provide rapid and easy access to information makes information management difficulties amenable to computer support.

4. Problems in analyzing or reasoning about the situation

Human decision makers often know how they would like to think about the problem, but find themselves unable to do so because of time and intelligence resource limitations. A person might realize that by exhaustively comparing a finite but very large number of options an optimal decision could be reached, but also realizes that in the time available the necessary calculations simply can't be performed. Often, such problems can be supported by some sort of computational algorithm.
5. Difficulties in visualizing

People tend to use visual representation in their decision-making processes, but they frequently have difficulties in manipulating these representations, particularly in a quantitative manner. At the same time, however, people are much better able to make such quantitative projections using a visual representation instead of a purely mental one. Using computers to provide concrete depictions of mental pictures can directly support the unaided decision process.

6. Quantitative inaccuracies in heuristic judgments

There are many situations where the human decision maker is required to make some inferences that only can be described as 'judgment calls'. Highly skilled and experienced decision makers can make such judgments with high reliability and consistency. But, when they have a numeric aspect to them, there is often a systematic bias in the judgment provided. Computers are unable to rival the judgmental abilities of humans, but they can be used to remove the biases. Computers are good at examining collections of past decisions to isolate the decision process and information used by the decision maker.

It is important to note that one or more of these six kinds of general decision-making difficulties might not be found in every given decision situation and that other general difficulties may be present in specific instances. Usually, a list of distribution-specific decision-making difficulties will include at most one process prediction aspect, one choice modelling aspect, and one problem presentation aspect, because a typical distributed decision-making situation involves one real-world process and has one and only one high-level goal. However, the information environment and decision processes can vary a great deal across distributed decision situations, so there may be a need for several kinds of decision support. Most decisions will involve several kinds of support needs, but only very few will involve all six kinds.

The six types of general decision-making difficulties described above are not the only ones that people have, but they are the most often
mentioned and common ones for which computer support technologies exist. Thus, six corresponding general support functions for human decision makers are listed below.

1. **process modelling:** computer-based models of real-world processes can be used to provide predictions of underlying processes at future points in time or under different conditions, compensating for human difficulties in process projection;

2. **value modelling:** mathematical formulas and/or rules for combining attributes of decision outcomes or making tradeoffs among competing objectives can be developed and automated, removing cognitive limitations in consistently applying value criteria;

3. **information retrieval:** computer-based techniques can be used to store, organise and retrieve data, information, and/or knowledge, extending decision makers' ability to access information in decision processes;

4. **automated reasoning:** mathematical, computer science, and artificial intelligence tools can be used to automate totally or partially key analytical reasoning steps used by expert decisions to compensate for situational constraints limiting the unaided decision maker;

5. **representation aids:** computer-graphics and/or natural language processing tools can be used to make data accessible to the decision-maker in terms of the decision maker's own mental representation of the decision situation; and

6. **judgment refining:** statistical techniques can be used to remove the systematic inconsistencies or biases that arise from certain quantitative heuristic human judgments.
These general functions are used in the subsequent chapters to organize the DSS technology base, providing a tool for DSS functional design. At this stage they serve as an intermediate step between the problem-specific decision support needs and functional design task.

3.6. Future possibilities

There are two main directions from which pressure is coming to overcome the shortcomings of existing information systems used for decision support: The business environment and users.

In the business environment, the increasing cost of personnel coupled with skill shortages is making it increasingly attractive to find ways to use advanced technology to provide more effective decision support. At the same time, increasingly aggressive national and international competition means that organizations have to adapt to a more rapidly changing environment. This leads to an increased need to use the expertise, special skills, and knowledge of an organization to gain competitive advantage (see also van Weelderen [46]).

Amongst military users, there is an increasing understanding of the capability of computers. As more and more soldiers are using computers, so they are coming to rely on them and view them as a necessary tool. The introduction of more and more powerful software tools has led to the expectation that increasingly sophisticated DSS will be become available and that these will be able to support a much wider range of decision-making tasks. At the same time, soldiers are faced with ever growing amounts of information which they have increasing difficulty coping with (see also Coppieters [47]).

There is a general need in organizations for flexible decision making structures. This requires timely information on events in and outside the organization. The search in the field of information systems for these structures can be described in terms of three noticeable trends. These trends have affected, and will affect even more the ways in which organizations design, build, and use information systems (Sol [48]).
1. Information technology can in principle supplement human performance.

Improving the performance of organizations by supporting human decision-makers is not done automatically by drawing on information technology. A methodology challenge must be taken up.

The advent of this new technology will result in the automating of elementary and routine operations. In return, this shifts the weight of user effort to a higher decision-making level where repetitive 'mechanical work' - see Panko [49] for definition of information work types - is replaced by tasks which coordinate lower level routines and which require decision-making and problem solving to cope with disturbances and variations in the work situation. A result of this tendency away from job specialization is the expanded need for information as well as what is called today an appropriate 'environment' (see Sol [35]) for generating, storing and manipulating this information. Today the information worker is becoming recognized as homo ludens (Sol [35]), a creative, bounded rational problem-solver that should be supported, rather than replaced.

2. Organizational decision-making is recognized as the primary organizational activity.

As the cost of coordinating decreases further, however, more decentralized organizational forms such as markets may become desirable (Malone and Smith [50]). Markets are assumed to have lower production costs, since they can further exploit economies of scale and average uncorrelated demand across many firms, but higher coordination costs, since more communication is necessary to find a supplier, and contracting with an outside party involves higher levels of risk and uncertainty. As the cost of coordination drops, these factors become less important. Organizational decisions may also allow physical decentralization of work. Many organizations are already geographically dispersed and are coordinated using telecommunications.

Another commonly discussed possibility is that the centralization of decision-making is inherently desirable to managers, and decentralization
takes place only because no one person can control the necessary resources (e.g., information, employees, soldiers) due to limitations in human information-processing capacity. These constraints force managers to delegate control over some decisions to focus on more important issues. The use of information technology may lessen these constraints in two ways, first by providing easier access to and facilitating more complete analyses, and second by providing a mechanism to control workers. The use of information technology may thus permit decisions to be made at a higher level and ensure their implementation by subordinates.

3. Information technology is used to gain a sustainable competitive advantage.

A potentially very useful way to examine the competitive advantages of information technology in organizations is to examine the kinds of information the organization uses and the ways it processes this information. Three basic assumptions underlying such an information processing perspective may be outlined: Organizations must deal with work-related uncertainty; organizations can usefully be seen as information-processing systems; and organizations can be viewed as composed of sets of groups. In this view, organizational decision-making influences the pattern and content of the information flowing between the subunits and the way they process this information.

Achieving a balance between these possibilities requires both the insight needed to be candid about the limitations on one’s organization and the leadership needed to withstand whichever pressure dominate at the moment. When a (dynamic) balance is reached, an organization can use its personnel most effectively and develop realistic strategies. When it is not reached, the organization is in a state of crisis, vulnerable to events or hostile actions that exploit its imbalances. The crisis is particularly great when the need for balance is not recognized or cannot be admitted.
Chapter 4

4. DEVELOPING A DISTRIBUTED DSS

The purpose of this Chapter is to guide development of distributed decision support systems in a military environment. It describes a methodology for system development; an approach to modelling a distributed system is defined and some design principles are specified. This Chapter also provides a description of model components for a DDSS.

4.1. A methodology for development

In the development of an information system, the operational requirements are defined first, then a system that fulfills the requirements is systematically synthesized. In many cases a system design language is utilized during such tasks. It is also an iterative process. In the next step, logical system architecture is established. Hardware and software components are determined, and the program design is finalized.

We shall now discuss several methodologies that have influenced the idea for the development methodology used in this study. To develop an information system in which more attention than usual was paid to the activities in a problem-solving process, Sol ([51]) introduced the use of simulation. The philosophy of this methodology was to be able to develop a dynamic model of the problem situation, experiment with this model, and experiment with alternatives for the problem situation. Several research groups have carefully been picking up the idea, and have worked it out in a number of different directions (Wierda [52]). Some of these approaches concentrate on the character of the system to be described, others view the modelling of information systems more from the point of the people who are involved in the development process (see Wierda [52] for bibliographies).

Another approach to develop information systems using visual interactive modelling (VIM) is reviewed by Bell in [53]. Visual interactive models may be used to help designers to incorporate 'human factors' in system development (Crookes, [54]).
Chapter 4

The term 'visual interactive model' is used in literature to describe a framework having three essential components (Bell, [53]):

1. A mathematical or symbolic model that derives optimum (or perhaps 'good') decisions to be displayed.

2. A computer-generated graphic display of a system or problem situation.

3. A user-friendly interactive interface that allows the user to run the model and display the results, and increasingly to edit the model.

The methodology derived for this study concentrates on human aspects because of the complexity of the decision-making in relation to Key Tasks that the DDSS under discussion is expected to perform. According to Boar (155) there is a growing body of evidence that suggests that prototyping has particular strengths for communicating and explaining human aspects in the results of modelling exercises to decision makers for problem formulation, model validation, and for the provision of support.

Prototyping must be as much reflected in states of mind as in a structured design methodology (Andriole [56]). Contrary to popular belief, prototyping is highly structured and the approach to it must be extremely methodical (Sol [57]). The primary assumption many prototype developers make is that interactive systems cannot be developed without input from prospective users. They assume that the system must be repeatedly developed.

The prototype system development methodology known by most of the practitioners and theoreticians is an iterative approach (Boar [59]). Figure 4-1 shows the prototyping process on an operational level. It starts with application selection and is quickly followed by some initial prototyping effort. The prototype as a result of these effort often serves as a combined feasibility study, design document, and a functional specification effort.
Figure 4-1: Prototype system development methodology

As the problem area becomes better understood by developers, more prototype efforts are undertaken with more complex implementation, testing and evaluation. The development team iteratively enhances each prototype until an operational system evolves. In the final development phase, the team is challenged to maintain and enhance the results of the prototyping efforts. The mature prototype must be transformed into an operational system.
Experience with this transition process (from prototype to an operational system) reported by (Reidel [59]) has shown that DSS development is necessarily an experimental activity. Supporters of this statement (Alavi and Henderson [59] and Carley [3]) contend that not enough is known about the domain or the technology to adopt a more standardized approach. Thus the field of modelling a DSS is experimental, DSS are not suited to design before implementation - the designs, and also the specifications, must be a result of iterative prototype building. Finally, since system specifications result from building each successive prototype, operational systems result from previous prototype enhancement efforts.

4.1.1. Transition issues

The difficulty of moving from a prototype to an operational system is common for any organization developing information systems. The prototype itself can prevent a transition to an operational system. Prototyping creates expectations for users and decision makers. When these expectations are not met, prototypes become antagonistic to further development efforts. They become not just flawed but dangerous.

Demonstrating a prototype to an audience affects everyone's expectations, and this can be dangerous whatever the system's virtues. Technically feasible prototypes of, e.g., DDSSs, lacking an appealing user interface, often look like 'unfriendly decision-makers'. They rarely have impressive front ends, and non-technical users can't understand the system's depth of reasoning. The audience is left distinctly 'underwhelmed'. Commanders refuse to accept a system that is so incomprehensible. The transition to an operational DDSS is never made.

Alternatively, prototype demonstrations that overwhelm an audience can also be dangerous. Some prototypes are aimed at exhibiting what will be the complete functionality of the finished operational system. In this demonstration, decision-support, reasoning, and representation issues are not solved but catalogued for future attention. For that portion of the prototype presented that will later make decision recommendations in operational use, only the user interface is implemented. Not every algorithm of the prototype can be used in the operational system. Only
the user interface is developed, and then presented as if the underlying algorithm were already implemented.

Commanders believe they see the complete system almost ready for use, and approve it for delivery. The developers may know the amount of work left to be done, but may be unable to express this to the audience in the face of the glorious prototype. When time and cost estimates for building the real system are much greater than expected, approval is rescinded. The transition to an operational system is never made.

A conclusion is that a prototype has to be explained thoroughly to users to provide them with a realistic view of the system as it eventually will appear, allowing them to relate what they see in the prototype directly to their requirements.

4.1.2. Towards a new methodology

An extended approach to the iterative prototyping methodology introduced in Figure 4-1 must with respect to the transition issues from the previous section rely on a more structured and phased process. The methodology proposed in this section details prototyping tasks, structures activities, and plans for deliverables. Although iteration is assumed between each task, iteration results are better described than in Figure 4-1. By dissecting the process into discrete jobs, iteration affects only a small area of prototyping without revising and redoing the entire project.

The extended methodology broadly divides prototype activities into two parts: Planning and development Figure 4-2. Prototype planning activities begin with specifying primary objectives of the prototype. The developers must know at the outset the extent to which the prototype is meant to discover technical feasibility or demonstrate functional form.
*) Reference point

Figure 4-2: An extended methodology for a DDSS-prototype development
Sub-problems are specified, and evaluation criteria determined. These criteria determine prototype success. In a decision-support system, the developer need to know what percentage of correct decisions (‘correct’ is defined as matching the commanders decision) will attain adequate payback for the system. Finally, a schedule and goals are set for the development process. Written specifications for development efforts result from the prototype planning phase.

During prototype development, tasks are divided into five stages: predesign, logical architectural design, physical architectural design, implementation, and evaluation. A product is delivered from each stage.

The predesign stage produces domain vocabulary and understanding. All domains have their vocabulary, which is relevant to the system. The predesign stage in a DDSS project for command and control, for example, may result in a document defining terms as ‘assessment report’, ‘situation map’ or ‘readiness status’. This is the first reference point for iteration in the prototyping process.

In the logical design stage, the reasoning and representation examples used in the domain are analyzed. The work product of this stage is a paper model describing the decision-making processes in that domain. In command and control, the logical design might represent phases of the C2-cycle, such as ‘maintain status’, ‘assessment’, and ‘planning’. The logical design document shows how these phases interrelate and provides a second reference point to control prototype iteration.

In physical architectural design, logical designs are translated for the hardware and software to be used. This is a critical step in the development process, but one that often must be redone in the transition to an operational system. For example, a prototype may be developed in PROLOG on a Macintosh, yet the operational system must be delivered in Ada on mainframe hardware with Macintosh front-ends connected via a local or wide area network. If the physical design has been separated from the logical design, as our methodology suggests, iteration must only regress to the logical design stage. Implementation estimates can be more accurate and increases in development budgets are avoided.
Finally, during the implementation phase e.g. operational data are loaded, and the prototype evaluation answers the questions set out in the planning phase.

Transition and deployment problems should be alleviated by using this methodology for system development. For example, if a change of hardware is required at deployment time, logical design documents remain largely intact. Explicit planning tasks also help to control expectations. With evaluation criteria determined before development, the prototype is less likely to be tested on issues it was not meant to solve.

Prototype efforts can and should be used as test beds to estimate the length and difficulty of the operational system development effort. As long as DDSSs remain new types of information systems, we will need prototyping efforts to explore ideas and unsolved problems. However, we need not despair that prototypes cannot necessarily address major problems in developing operational systems. When development teams know that prototypes will make the transition to an operational system, they should structure their efforts to solve deployment problems. By tracking prototype development information, restricting the purpose and audience of the prototype, and focusing efforts, DDSS builders may largely avoid transition problems.

After introducing the new methodology of developing a distributed DSS, we will now describe the DDSS design process and present some design principles.

4.2. Modelling and design principles

This section addresses the design of a DDSS: It seeks to combine classical concepts (Andriole [56]) with new ideas in a framework which permits the study of distributed decision problems in a meaningful setting.

Because of the broad scope of this concept for the design and the complexity of the problem to which it is addressed, the description of the design is organized into two parts. This section presents the background against which the design is discussed and some principles which can be
usefully exploited in understanding the design. The description is completed by incorporating these principles into the strategies for coordination of the activities of individual parts of a DDSS and the discussion presented in Chapter 6.

Some approaches to the design of a DDSS already exist, as shown below. Some of these approaches break down the procedures for decision support on an existing DSS and implement them in a distributed manner (Gallager [60]). Other approaches accept a model of a DSS and modify it to meet a distributed solution structure. These approaches involve modification techniques such as time-scale separation, interaction separation, chaining, aggregation and other application-specific techniques. The modified model is used to derive a design strategy that matches the distributed structure (Witsenhausen [61] and Barta [62]).

The conclusion is that the distributed system model should be designed without initial recourse to a centralized model.

In general, this conclusion tackles the problems of distributed decision-making using the fundamental assumption that no decision-maker has a complete model of the system. Each decision-maker has knowledge of the operation of a particular cell - he is an 'expert' on this subsystem. He has nearly no knowledge of the structure of the system outside of his domain; assessment of impact of his decisions on the rest of the system and of external decisions on his subsystem must be gained through communication. The coordination of planning activities is highly dependent on available communication resources.

The study of how existing organizations actually go about decision-making has been spurred by the goal of improving that process. One major conclusion (Simon [63]) is that organizational structure has a significant impact on the decision process. Alternatives that a decision-maker might consider do not occur to him if they are outside his scope of work or involve knowledge to which he has no access.

However, the models employed by all decision-makers in a distributed system must in some way describe the entire organization, and each
decision-maker must be made aware that other parts of the organization exist and affect the portion of it that has been assigned to him. So the first principle of an approach to design distributed decision-making in a military headquarters is suggested.

**Principle 1:**
Each commander possesses a limited model of the headquarters under his control.

We consider two such commanders such that the decisions of one (A) directly affect the decisions of the other (B). Clearly there is a need to communicate. B needs to advise A of the decisions he has taken so their effects can be accounted. A needs to inform B of his objectives so B can plan actions to help A achieve them. However, care must be taken to prevent strategies where an infinite bandwidth channel is used to communicate everything to a single side.

If two commanders have almost completely separate models, the set of subjects which they might communicate is naturally limited. All that two interacting commanders might share is the set of interaction messages produced by one which affect the other.

If this set of messages should be the only common context which they have as a basis for communication, a second principle is suggested.

**Principle 2:**
Inter-commander communication takes place only in terms of quantities directly related to the underlying interaction messages.

A common approach to distribute messages between decision-makers is to use iteration between them. While often effective, this usually requires tremendous communication capacity and also security considerations as several iterations have to be made at each time step to determine a set of current control inputs. For this reason, the strategies shall operate as follows.
Chapter 4

Principle 3:
Avoid iterative techniques that involve communications between commanders at each step.

As a result, mapping the real world of a headquarters into a model includes the fact that commanders will often be ignorant of many things going on which might eventually affect them. A probabilistic approach to such uncertainty is not yet well defined. It seems preferable to resolve the uncertainty by assuming the worst case. This has the appealing advantage of granting local autonomy to decision-makers. Communication helps to generate agreements between two commanders restricting what each commander will do. However, each commander can be free to choose one of the several alternatives within the agreed restrictions, knowing that the other will prepare for what is seen as the worst case possible.

Principle 4:
Uncertainty about a commander's future decisions will be resolved either through communication or through assumption of the worst case.

This principle is particularly true for military headquarters, as 'military training' advises commanders to assume the worst case to minimize any kind of risk.

4.3. Modelling distributed decision-making

Following the principles introduced above, the goal of modelling distributed decision-making (DDM) is to provide the essential components of distributed decision-making models and support algorithms. Besides describing the nature of the models, a description of the algorithm structure is necessary.
4.3.1. Description of model components

The description of the DDM domain is by means of a directed graph (see section 6.2.1)

\[ \Phi = (V, A) \]

where \( v \in V \) represents a component (node, cell) of the DDM domain and the edges \( a \in A \subseteq (V \times V) \) represent the neighbour relation. The neighbour relation is used to model the interaction between a pair of DDM elements. It defines the set of pairs of cells that have an opportunity of passing messages. Multiple message exchanges are required for communication between a pair of DDM algorithm components not defined in the neighbour relation.

The neighbour relation \( N \) is formally defined for a cell \( v_i \) as

\[ N(i) = \{ v_j | (v_i, v_j) \in A \} \]

This definition allows specification of the extent to which global information (regarding the states of neighbours and non-neighbours) is involved in decision-making. In this way, we arrive at a description of information flow and the limits on the amount of global information directly available to each cell.

4.3.2. Description of DDM algorithm structure

The characteristics chosen here to describe a DDM algorithm are structure and semantics instead of syntax and semantics. Use of the concept of structure is justified on the basis that for example, the concept of syntax yields only the form, and not the content. The structure of a DDM algorithm can also be depicted as a lattice which supports the semantics of an algorithm. The structure of normal sequential algorithms would, for example, embody the underlying data structure. Analogously, the structure of a DDM algorithm contains:

- topology of decision-making entities,
some representation of static global knowledge at each node, such as the number and location of other nodes in the system.

Semantic characteristics are those which describe the actions of the algorithm during its execution and the effect that the distributed components have on their environment.

4.3.3. Description of semantics

The semantics of a DDM are represented by finite automata (FA) at each node. Each node of the graph Φ represents a cell and these cells communicate in a manner defined by the neighbour relation. Communication of state information between cells is accomplished through the input and output components of each cell. The output of one cell is input to its neighbours and represents the exchange of information between cells. Therefore, a basic unit of measurement describing efficiency\(^4\) is defined by the components (messages) of finite automata output.

The 'phase' of a cell at any particular time is an element of the parameter set of state and output at that time. The cell phase is used to define state transitions. Cell phase is therefore related to C2 cycle phases. Phases are used to characterize equivalent classes of cell states, to simplify specification and reduce possibilities of erroneous strategy modelling. A change in cell phase may occur on receipt of a message from a neighbouring cell. Another primary use of phases is to allow precise definition of global information used by distributed components in a decision-making system.

There has been prior use of the notion of FA (Brand [64], Malm [65]), primarily in the analysis of communication protocols (Aho [66]). The modelling described here differs from each of these examples by the use of phases, and in the way in which the discrete passing of messages is defined.

\(^4\) Efficiency. Taking action with the least expenditure of resources.
The formal definition of an FA in the cell \( v_i \) is as follows:

\[
v_i \ (Q(i), \Sigma(i), \Delta(i), \delta_i, s_{oi})
\]

where

\[
Q(i) = Q_{int}(i) \times Q_{ext}(i) \times P(i) \times P_r(i)^k; \quad k = |N(i)|
\]

\[
\Sigma(i) = P_{rin}(i)^k \times Q_{ext}(i)^k; \quad k = |N(i)|
\]

\[
\Delta(i) = P(i) \times Q_{ext}(i)
\]

\[
\delta_i: Q(i) \times \Sigma(i) \rightarrow Q(i)
\]

\[
s_{oi}: \text{initial state of } v_i
\]

and

\( Q_{int} \) represents the portion of current state which is internal to a cell and is not available to the neighbours of \( v_i \).

\( Q_{ext} \) denotes the portion of state which is exchanged between neighbours.

\( P(i) = \{P_0, P_1, \ldots, P_n, \ldots, P_m\} \) represents the current phase of the portion of the algorithm module at \( v_i \). Phase is introduced to allow precise specification of the amount of global information used in making decisions, and is discussed in more detail below.

\( P_r(i)^k \) is \( v_i \)'s current record of its neighbours' state. It represents the current phase of the portion of the algorithm module at \( v_i \).

\( P_{rin}(i) = \{P_0', P_1', \ldots, P_n\} \) is the actual current state of the neighbours of \( v_i \); the input from a neighbour to \( v_i \).

\( Q_{ext}(i) \) represents the last transmitted value of \( Q_{ext} \) from each neighbour of \( v_i \).

The superscript \( k \) in \( P_r(i)^k \) denotes a vector of \( k \) elements, \( P_r \), at cell \( i \).

The portion of the model which transforms one state of a cell into another is the transition function \( \delta \). This function maps current state and current input into the next state. There is no restriction requiring each cell
Chapter 4

to have the same transition function. For each cell the interaction is defined for two basic types of state transitions. The first state transition is a result of a local phase change, implying that a new set of local decisions has been made and transmitted to all neighbours. The second is in response to the phase change of some neighbour. Hence, the components of input $\Sigma$ are precisely the components of output $\Delta$ of the set of all neighbours of a cell.

Information exchange is controlled in several ways. First, the phase concept taken from the Command and Control cycle is used to characterize the exchange of information between cells at discrete points in time. This is a distinguishing feature of distributed systems because information is sent periodically between cooperating elements only when the sender chooses. A change in phase at one cell triggers a change of state at a neighbour since only at this time does a cell update $Q_{\text{ext}}$ to reflect a change in the component of state which is shared with neighbours.

The subscripts of the elements of $P(i)$ are related to the phases of the C2 cycle and are used to denote the degree of information exchange using the following conventions:

$$P_1 \rightarrow P_2 \rightarrow P_3 \rightarrow P_4 \rightarrow P_5 \rightarrow P_1 \rightarrow \ldots$$

$P_5 \rightarrow P_1$ represents the fact that a decision (phase 4) based on the accumulated information from the first three phase transitions has been carried out.

The last transition is necessary to allow the algorithm to maintain the level of autonomy for a commander. For example, if the set of allowable decisions includes those that permit a commander to attempt to exercise control over the resources of a neighbour, then additional phases are required to determine whether the neighbour will permit such actions; eg, allowing this type of decision does not imply that they will be irrevocable in all cases. Additional phases may also be defined to guarantee consistency and agreement among neighbours.
4.3.4. Summary of modelling features

Features of the introduced model are:

1) Separation of internal and external status information.

2) Introduction of a phase concept for the classification of the information flow between the cells and for the identification of the messages needed for individual decisions.

3) Definition of non-homogeneous transition functions for any particular cell.

In practice, the components of the model can be modified to such an extent that they apply to many DDM problems. This applies particularly to the case of degraded communication between cells. However, in such cases knowledge of the phase status of a cell and of the information flow between cells is not always available.

Since the flow of information between cells is made up of individual, separable messages, introduction of the phase concept allows accurate definition of individual cell status. If the number of messages which are exchanged until phase-change of a cell is known, a statement can be made about the volume of information necessary to arrive at a particular decision.
Chapter 5

5. REQUIREMENTS ANALYSIS OF DECISION-MAKING PROCESSES

This Chapter starts the description of the development of a prototype DDSS by discussing requirements analysis. An approach for requirements analysis based on logic programming is suggested that facilitates iteration throughout the subsequent design and implementation stages as shown in Figure 5-1. The analysis is demonstrated for Key Tasks.

5.1. Introduction

Written requirement documents for decision-making in military headquarters contain an inventory of standardized Key Task statements. To resolve ambiguity and facilitate automation, it is desirable to obtain an executable representation of these documents.

Symbolizing the natural language text into a logic programming language can provide this executable representation without altering the pertinent semantic relationships expressed. This approach was proposed by the Logic Programming Group at Imperial College, London, who symbolized most of the British Nationality Act into PROLOG, demonstrating the iterative nature of the symbolization activity, and concluded that the resulting formalization would support the mechanized analysis of the logical consequences of the act (Sergot [67]).

The product of a requirements analysis should be a specification that describes the requirements as they are needed for design and implementation of distributed decision support systems. Several tools are available that should support the generation of such a specification, including PSL/PSA [68] and SREM [69], but these tools presume that the requirements have to be expressed in a special-purpose language.

One of the better known analysis and specification methodologies is the structured analysis of DeMarco [70], who recommends a combination of a data flow diagrams, ‘minispecs’ and a data dictionary to describe the requirements of a system. An attempt is made to express these three components of a structured specification in a logic programming language,
following the approach introduced in the beginning of this Chapter. Also, an attempt is made to express much of the expertise needed in performing requirements analysis as a set of rules in the same logic programming language.

An existing tool which implements some features mentioned above is 'The Analyst' providing PROLOG-based automated support for requirements analysis (Stephens [71]) and employing a methodology named CORE (COntrolled Requirements Expression).

Input to 'The Analyst' are requirements in CORE format, and a knowledge base of PROLOG facts is then compiled to represent them. The CORE methodology is represented by a knowledge base of PROLOG rules, which helps to identify errors in the requirements. The output of 'The Analyst' is a specification in CORE format with underlying PROLOG representation.

Unlike 'The Analyst', our approach to requirements analysis described in this Chapter is not based on a specific format. Once the written requirements, e.g. the Key Tasks, have been symbolized using PROLOG, a knowledge base of rules about the specification domain provides an initial indication of what appears inconsistent or incomplete, and is used to generate error reports providing the opportunity for problems to be resolved.

Another knowledge base of rules about a graphical presentation technique of structures should transform the requirements into an early version of a structured specification, while the production of trace reports should make it possible to keep track of what requirements are being addressed.

The different steps for requirements analysis using PROLOG as the implementation language, are illustrated in Figure 5-1. The application of this approach to analyzing Decision Key Tasks is demonstrated.

The first activity, 'Symbolize', expresses the Decision Key Tasks as PROLOG facts using predefined vocabulary of PROLOG predicates. The
remainder of the approach may be thought of as a progression of PROLOG facts, starting with the 'Analyze' activity to locate errors.

Evaluation of these errors by the 'Interpret' activity closes the loop by correcting the PROLOG version of the requirements. When the errors have been removed the requirements are transformed by the 'Specification' activity, whose output defines a set of data flow diagrams with which the design can be worked out and where trace reports can be extracted by the 'Trace' activity.

Figure 5.1: Approach to requirements analysis by logic programming
The examination of trace reports by a developer may lead to modifications to the PROLOG version of the requirements.

The following sections explain the five activities of the approach.

5.2. Main activities

5.2.1. Symbolization

The 'Symbolize' activity has to be done manually, and involves parsing the natural language text of the written requirements (e.g. Key Tasks). Due to the need to provide clear guidance for the decision maker, natural-language documents for Decision Key Tasks are normally written in short sentences, which can easily be parsed with grammar rules. Therefore, it seems likely that the semantic problem in automated symbolizing can be avoided, and that natural-language processing may eventually replace the manual process.

'Symbolizing' is done in this approach by extracting the pertinent relationships using simple declarative sentences, and expressing each sentence as a PROLOG clause using predefined PROLOG predicates.

A limited vocabulary of predicates must be generated before symbolizing for each functional area\textsuperscript{5}, taking account of the entities and relationships likely to occur. This set of vocabulary of predicates evolves until it is adequate for symbolizing the requirements of any Decision Key Task in that functional area.

Most of the predicates are common to a range of functional areas, because many terms are common to the natural-language description of the Decision Key Tasks of different functional areas. The less common terms appear as arguments instead of as predicates. The resulting PROLOG clauses will be called the decision facts, and are entered into the

\textsuperscript{5} Functional Area. Generic areas (e.g., intelligence, nuclear operations) within a war headquarters (WHQ) that may involve any number of WHQ organizational elements (cells/sections/divisions) and may consist of any number of Key Tasks.
requirements knowledge base.

The following rules, which are part of a grammar for Decision Key Tasks defined in PROLOG, were used to symbolize and parse Key Tasks.

Example 5-1:

task --> imperative, object, det_object1, det_object2, nmbr.
det_object1 --> determiner, object1.
det_object2 --> determiner, object2.
imperative --> [approve].
imperative --> [decide].
imperative --> [determine].
imperative --> [direct].
imperative --> [authorize].
imperative --> [formulate].
imperative --> [prioritize].
imperative --> [decide/approve].
imperative --> [approve/validate].
imperative --> [approve/prioritize].

The --> symbol is a standard PROLOG operator. It is defined to convert DCG (Definite Clause Grammar) clauses into regular PROLOG clauses.

5.2.2. Analysis

The requirements facts placed in the requirements rule-base by the 'Symbolize' activity are first analyzed by a PROLOG implemented application-specific rule base containing rules on the Decision Key Tasks. The rules are designed to operate on the vocabulary of predicates generated for a specific functional area.

The objectives of the analysis are to discover cases of inconsistency and incompleteness in the requirements and to produce error reports for the
Interpretation activity. The Analyze activity examines the structure of requirements facts, and produces the baseline requirements facts from which the specification will be prepared.

The following predicates are used in the ‘Analyze’ activity to display the structure of the requirements and to identify errors.

Example 5-2:

\[
\text{In}(\text{DKT}, \text{SKT}, B)
\]

Entity DKT (Decision Key Task) has input from a source SKT (Supporting Key Task), and requirement reference B.

\[
\text{Out}(\text{DKT}, \text{SKT}, B)
\]

Entity DKT has output to a destination SKT, and requirement reference B.

\[
\text{Level}(\text{KT}, M)
\]

M is the number of ancestors\(^6\) of Key Task KT.

The rules for analysis are contained in the application-specific knowledge base, which controls the ‘Analyze’ activity. Any error reports produced by the ‘Analyze’ activity may modify the requirements facts used to produce the baseline requirements facts. The following predicates serve to illustrate the detection of inconsistency and incompleteness.

Example 5-3:

\[
\text{mklist}(X) : -
\]

\[
\text{ktask}(X),
\]

\[
(\text{phrase}(\text{task}, X) \rightarrow \text{true};
\]

\[
(\text{nl}, \text{write}(X), \text{write}(\text{'} \text{is part of a Key Task with an invalid structure'})
\]

fail.

---

\(^6\) The ‘ancestors’ of a Key Task are all those higher level Key Tasks (with respect to the Command and Control cycle) to which it contributes.
**Chapter 5**

\[mkrel(X) :-
rel(X),
(phrase(support,X) -> true;
nl, write(X), write(' subtask is part of a Key Task which is
neither a Decision Key Task nor another subtask'),
fail.\]

The first predicate checks whether a list of Key Tasks conforms to the
definition of a valid Key Task and indicates those with invalid structures.
The second predicate indicates an inconsistency if X is a subtask of a
Decision Key Task ‘task’ which does not exist.

The output from this phase is an error report which is evaluated by the
‘Interpret’ activity.

**5.2.3. Interpretation**

Another manual activity, ‘Interpret’ involves determining which errors
reported by the Analyze activity should be acted upon, and provides any
corrections for the Symbolize activity to modify the requirements facts.

It is not necessary to remove all ‘errors’, as there may be intentional
incompleteness reflecting a ‘don't care’ decision by the user, and offering
flexibility to the developer. For example, a Decision Key Task may be
decomposed by the user into two lower level tasks, with no requirement
as to how the two are related. The developer can later identify the
interrelationships between them by specifying what data they exchange.

**5.2.4. Specification**

The baseline requirements facts are converted into structured specification
facts with the aid of a technique-specific knowledge base. They may
allow application of presentation techniques such as the
Yourdon-DeMarco methodology.

A data flow diagram is a network of activities that transform input data
into output data. Each activity may be partitioned into lower-level
activities, down to any level of detail. Supporting descriptions are required
to define the data that appears in each input and output (data dictionary),
and the transformations performed by each activity ('minispecs').

The 'Specify' activity includes the allocation of baseline requirements facts
to a set of data flow diagrams. The associated minispecs and data
dictionary completing a structured specification are omitted in this
discussion.

Each component of the data flow diagram is defined by one or more
PROLOG facts, which together represent the structure of the requirements
as well as the graphic primitives.

Each PROLOG fact references all the paragraphs in the written documents
which led to the need for that component, so that the resulting
specification knowledge base includes the audit trail required for trace
reports.

The following predicates are used in the Specify activity to define
components of a Yourdon-DeMarco specification.

Example 5-4:

Member(X,Y)

    Used to define a member X of a List Y.

Diagram(N,L)

    A drawing with name N and level L.

External(N,D,C)

    A box with name N on diagram D with requirement reference C.

Process(N,P,C)

    A bubble with name N on diagram P with requirement reference C.
Chapter 5

Dataflow \((L, S, D, C)\)

An arc with name \(L\), source \(S\), destination \(D\) and requirement reference \(C\).

These predicates include one which will be used to define a PROLOG construct (member), one which will be used to generate individual data flow diagrams according to the conventions of structured analysis (diagram), and three that are involved in the actual modelling of the requirements being analyzed.

5.3. Conclusions

The simple examples given in this Chapter serve to analyze the domain of Decision Key Tasks, but the effect of such an approach on the development of large complex systems is more than expected. A logic programming language such as PROLOG can be used to represent the relevant requirements of decision making in any domain in analyzable form right from the beginning of a development, facilitating the mechanization of the entire lifecycle from requirements analysis to software maintenance.

The benefits include the possibility of detecting more errors in the requirements-analysis phase, where they typically cost an order of magnitude less to correct than in the design phase. Also, the first version of a graphic form of the specification, which is the form usually preferred for manual analysis, can be produced immediately after the requirements documents have been symbolized. Finally, any changes to the requirements (or to the application or technique rules) immediately ripple through to the specification, avoiding delays which might affect the entire development.

While symbolization can be done manually, steps towards automating this activity are available on the basis of recent progress in natural-language processing.

Existing programs for generating data flow diagram layouts from a description of their components (processes, data flows, externals,
connectors, and stores) may be adapted to PROLOG to automate the 'specify' activity.

An example of requirements analysis in relation to a Decision Key Task is shown in 64.
6. LOGIC-BASED MODELLING OF DISTRIBUTED DECISION-MAKING

This Chapter describes a model of a DDSS, which consists of a database, a modelbase, and a user interface. The requirements analysis that was discussed in Chapter 5 leads to the contents of the database component (section 7.4). This Chapter concentrates on the modelbase and outlines the user interface. In the next Chapter these three model-components are used to specify the architecture of a DDSS.

The modelbase description encompasses the definition of finite automata (FA) (section 4.3.3), directed graphs, and the discussion of various strategies for coordination of cells within several kind of organizational structures. An algorithm for solving a class of temporal reasoning problems that can be employed in all cells of an organization is created.

6.1. Model description

The three central model-components of a decision support system in the taxonomy of Sprague and Carlson [72] are: A database management subsystem, a modelbase management subsystem, and dialogue generation and management subsystem. These subsystems need to be integrated and interrelated as they interact together to affect distributed decision-making. The database component includes both internal and external data. Internal data are either transaction data or data collected internally from other subsystems. Associated with the database is software called the DBMS (DataBase Management System). This software creates, modifies, and maintains the database as required by the user. The database management component should enable a DDSS to perform any type of data analysis operation.

The model base component includes a series of mathematical or logical algorithms, which - in conjunction with the database - should enable a DDSS to operate in any type of organization with structures that are discussed later in this Chapter.
Finally, the dialogue management component provides the user with different interface procedures that enable him or her to access the DDSS and is, from the user's point of view, probably the most important part. An extended example for an user/system interface is provided in Chapter 8. At this point we provide a rough overview on different kind of user/system interfaces.

Once the DDSS has been built, its usability depends on the quality of the user interface. Figure 6-1 provides an indication of the range of possible interface facilities. In most information systems the user interface is often designed to allow interactive dialogue. This dialogue and the initial input most often appear to the user as structured data-input arrangements incorporating menu choices.

Other capabilities often found in information systems are user interfaces that allow the user to select alternative parameter values and observe the effect on the outcome. Facilities are also found that allow the user to perform an initial pruning of the sequence of questioning (through menu choices) so that the system need not pursue areas that the user feels are irrelevant or unnecessary. Another facility of user interfaces may be the capability to save examples for future use. Information systems may also include user interfaces with interactive graphics and simulation algorithms that increase the end-user's understanding and control of the system being represented.

Above all, the user interface must be user friendly if the system is to be accepted.
6.2 Coordination strategies for distributed systems

In this section, strategies based on the underlying interactions between cells of an organization are developed to coordinate decision-making. A number of strategies using varying grades of communication to reduce the uncertainty concerning the events controlled by others in each commander's decision-making is presented. The organizational requirements of each coordination strategy in terms of constraints on the cell relationships are derived. It is shown that any of the strategies discussed here work well in hierarchical organizational structures, and some are also adaptable to non-hierarchical type of structures.
6.2.1. Definition of structures

The information flow in an organization (e.g. military headquarters) is determined by the pattern of interactions between individual cells and specified by the neighbour relationship $N(i)$. Such structures are never simple, but exhibit complex branching (see the example in section 6.4).

Information flow may be represented by directed graphs. Structures of such a directed graph $\Phi$, with cell set $V$, are defined as follows.

**Definition** $\Phi$ is singly connected if, for any $v_i, v_k$ distinct cells $i, k$ in $V$, there is exactly one ordered subset on $V$, $(v_i, \ldots, v_k)$, such that

1) either $(v_i, v_{i+1})$ or $(v_{i+1}, v_i)$ is in $\Phi$, for $i = 1, \ldots, k-1$;
2) $v_i \neq v_j$ for any $i, j$.

This requires that there be exactly one path from each $v_i$ to another $v_k$ where the path can follow interactions in either direction. The nodes in such a directed graph may be partitioned into three sets. For example, superior, inferior, and middle cells are labeled $S$, $I$, $M$, respectively.

![Figure 6-2: Singly connected topology](image)

72
These sets are of interest as the decision that is made in a cell will depend on its relation to the rest of the organization as captured by the above definitions. It may be interpreted from the flow of interactions from inferiors to superiors that the inferiors produce interactions to fill the needs of the superiors.

Another concept relevant to directed graphs is that of an induction relation that orders the cells in some way.

**Definition** An induction relation $H$ on the cells of a directed graph is a relation that satisfies the following two conditions:

1) If $(v_i, v_j)$ is in $\Phi$, then either $(v_i, v_j)$ or $(v_j, v_i)$ is in $H$, but never both.
2) For any $v_i$, there is at most one $v_j$ such that $(v_i, v_j)$ is in $H$.

$H$ provides a way of working through $\Phi$ so that each cell is traversed once. It is an immediate consequence of the definitions above.

**Definition** A hierarchical system is one in which no cell affects more than one other. For each $v_i$, there is at most one $v_j$ such that $(v_i, v_j) \in A$. ($A \subseteq (V \times V)$, see also 4.4.1). Thus it is singly connected.

**Definition** A system is doubly connected if it contains a singly connected system and can be constructed from that system by adding only links $(v_i, v_j)$ in relation to $(v_j, v_i)$ to that system. Therefore if $A_2$ is the interaction relation for a doubly connected system, then there exists $A_1$ such that $A_2 \supseteq A_1$ and $A_1$ describes a singly connected system, and

$$A_2 \subseteq \{ (i,j) | (i,j) \in A_1 \text{ or } (j,i) \in A_1 \}.$$ 

A doubly connected system allows bidirectional flows of interactions between cells. Such systems can, for example, be used to model structures such as markets.
Figure 6-3: Doubly connected topology

**Definition** An acyclic system is one such that for each link \((v_i, v_j)\) there is no sequence of links starting at \(v_j\) which leads back to \(v_i\). Equivalently, there is no pair of distinct cells \(v_i\) and \(v_j\) such that both \((v_i, v_j)\) and \((v_j, v_i)\) are elements of the transitive closure of \(A\).

Figure 6-4 shows a conceptual view of one possible structure of an acyclic system. From this structure, we can deduce that cell 7 is in direct relationships with cell 2, cell 5, and cell 6, and that the cells 1, 3, 4, 8, have an indirect relationship with cell 7.
Figure 6-4: View of an acyclic system

Finally, an irregularly structured system is one with no restrictions on \( \Phi \).

6.2.2. Different coordination strategies

Neighbour relationships define connections between finite automata (FAs) in one cell and those in other cells. The relationships of the FAs to incoming and outgoing cell information and to organizational objectives require more detailed consideration.

**Inputs:** Each message received in a cell may be processed by only one commander, who has responsibility for deciding the immediate
consequences of receipt of the message.

**Outputs:** Every message leaving a cell must also be processed by just one commander, who has responsibility for deciding its content in accordance with his understanding of the overall system with regard to the coordination strategy in operation.

**Objectives:** Certain commanders will have precisely defined objectives in relation to the system as a whole. Their decisions are based on an understanding of the overall system influenced by the messages received. Other commanders have the responsibilities of coordinating cooperation between the remainder, that the understanding of the system as a whole by the latter may be improved.

In a singly connected topology one commander's decisions affect another commander's cell via a single series of cells. In this section four different coordination strategies for this topology, proposed by Tenney and Sandell in [73], are assessed and modified. These modified strategies are considered from the point of view of their applicability to other than singly connected topologies.

Each of the following coordination strategies is discussed for a particular degree of communication and follows three principles:

1) Coordination between cells is undertaken on the basis of choice of messages needed, in the opinion of the sender, for a decision to be reached in the receiving cell. The resultant decision problem in the receiving cell is solved independently.

2) Predictions about possible consequences are restricted to those most necessary.

3) Decisions as to when there is uncertainty are reached by allowing autonomous decisions by the cell commander.

Each commander is responsible for scheduling his own resources and for making decisions regarding the commitment of his resources to the use of
other commanders in the system. Therefore, each finite automaton (FA) in a cell normally consists of several computational procedures responsible for supporting the interests of the commander.

Each cell $v_i$ is defined as in section 4.43, and refined as follows:

$$Q(i) = L_{\text{int}}(i) \times L_{\text{ext}}(i) \times P(i) \times P_r(i)^k \times D_r(i)^k; \quad k = |N(i)|$$

where

$L_{\text{int}}(i) = \{0, 1, 2, \ldots\}$ represents the actual internal load of $v_i$.
$L_{\text{ext}}(i) = \{0, 1, 2, \ldots\}$ is the last value of load which was sent to neighbours during previous phase change.
$P(i) = \{p_n\}$ is the current phase of $v_i$.
$P_r(i) = \{p_0, p_1, p_2, \ldots, p_n\}$ is the last known state of neighbours of $v_i$.
$D_r(i) = \{\ldots, -1, 0, 1, \ldots\}$ is the set of decisions of the neighbours of $v_i$ with respect to $v_i$.

The first strategy considered relates to a static coordination problem. Since making the coordination problem dynamic results in major difficulties, the case of complete communication when each commander takes only one decision will be considered first. The second strategy relates to the case in which there is no communication. The third strategy relates to complete communication but only within a limited future period. The fourth and last strategy deals with the case in which only a priori communication is possible up to a certain point in time and after that point only forward transmission of messages can be guaranteed.

The following notation is used in the discussions below:

- $a^+ = a$ at some future time,
- $a^* = \text{optimal value of } a$,
- $\bar{a} = \text{sequence of } a$'s.
Chapter 6

A. Static teams

In a static situation, each commander can be viewed as making only one decision. In a distributed scenery the formulation of the coordination problem must be performed so that the decision of one commander can affect the situation of another, or the need for communication vanishes. In a singly connected system the options for such impact are restricted, but impose a constraint on the order in which decisions must be made by various commanders if all possible interactions are to happen. The first decision must be made by a subordinate commander, the last by a prime commander, and the decision times $t_i$ of commander $i$ is set to be $t_i = t_j + 1$ for all $j$ such that decision $D_j$ affects decision $D_i$. Such an ordering is clearly possible for any singly connected system. With these $t_j$ the static team decision problem is to minimize the system load in terms of message overhead $c_i$ to achieve a coordinated selection of the $d_i$ (compare to definition of efficiency given in 433).

$$\min_{d_i} \sum_{i=1}^{T-1} c_i(P,L,D)$$

For a commander to make an 'efficient decision' in the sense defined above, he must know the message overheads of the policy employed incurred by the rest of the system in producing or responding to interactions.

Static team problems therefore can be solved in an optimal distributed fashion in the case of full communication. The next strategy will introduce a way of coordinating distributed systems in the other extreme case: Using uncertainty to overcome a total lack of communication capability.
B. Coordination with no communication

This section addresses the question of making decisions in an environment where commanders cannot communicate. Each can only seek to minimize his local contribution to total system load, with no way of anticipating other commander's decisions or gauge his effect on others. Because of the uncertainty regarding interactions affecting it, each commander may adopt the minimum-maximum philosophy for making a decision.

Reflecting these considerations, the decision problem for $D_i$ is given for all $p_i$ and all $l_i$ (from the assumption of perfect knowledge of current interactions) find $d_i$ subject to local cell FAs.

$$\min_{d_i} \max_{l_i} \min_{d_i} \sum_{\tau=1}^{T-1} c_i(p_i(\tau), p_i(\tau+1), l_i(\tau), d_i(\tau))$$

Note that the minimum-maximum over $l_i$ and $d_i$ must be interlaced properly to reflect the fact that each choice of $d_i$ is made with knowledge of the current and past values of all $l_i$.

This strategy applies to systems where each commander knows nothing about the system outside his local model, either a priori or in real time. It requires the use of uncertainty and minimum-maximum decision-making, which in turn, leads to suboptimality in the team sense. The next strategy presents ways to improve overall performance by reducing the uncertainty under which each commander must plan.

C. Limited-horizon coordination

In this section, a more effective form of coordination than B is discussed where communication is bidirectional and is done by producing or responding to possible sequences of interactions over a finite period. The approach is to select the best general interaction for some fixed horizon $T$, given total uncertainty concerning interactions after $T$. This requires a way to compute the system load to produce or to respond to each interaction sequence by each cell of the system (as ordered by $H$).
Chapter 6

It is required that

1) each commander computes messages-to-produce or messages-to-
respond values based only on locally available information,

2) missing information is assumed to take its worse case value, and

3) each commander can update his plans at each time step to include
new coordination information.

For every decision problem $D_i$ the cell commander receives a $\tilde{z}^*$ from its
successor indicating the selected interaction sequence. The strategy at
subject then finds the argument that minimizes the costs from
predecessors affecting it, and costs from those affected by it assuming the
received $\tilde{z}^*$ is produced (or is to be produced), sending the resulting $\tilde{z}^*$'s
to appropriate predecessors and applying the first element of $\tilde{u}^*$ (local
decision set) as its control. Since this strategy allows the effects of one
commander's decisions on his neighbours to be included, albeit for a finite
horizon, it is to be expected that it leads to a better efficiency than that in
strategy B. This can be shown through the argument that the trajectory
selected by the min-max strategy is always available for selection by the
finite-horizon strategy, and another strategy is selected only if it is known
to be better, in a global sense, than that min-max decision.

There are two difficulties which can be encountered with the finite
look-ahead strategy. These are:

1) Communication load grows exponential with the length of horizon
(unless the organization has a special structure).

2) Horizons that are somewhat short often do not give much
improvement in efficiency. This suggests that an improved evaluation
on the relation between the length of the horizon and communication
load should help to shorten the necessary horizon greatly.
D. Coordination with abstraction

Turning from strategies that use increasing amounts of on-line communication to achieve improved efficiency, this strategy draws on the concept of abstraction to provide each commander with a simplified model of the external system that can be used to reduce uncertainty, either on the incoming interactions or on the way decisions may lead to increased information load in the other cells.

The procedures embodied in this strategy will use the induction relation $H$ to construct abstracted models (possible interaction sets) of successively larger sections of the organization until the terminal cell has a single, simplified model of the entire organization. Each commander receives abstracted models from his predecessors (e.g., possible responses to a situation assessment), constructs a hybrid model by adding his set of possible information transfers, (e.g., alternate plans, state information, etc.), and then passes an abstraction of the hybrid to his successor. Then, using no further communication, the abstraction can be used to reduce uncertainty as to when interactions will be produced by inferiors. This is because interactions are similar to observations, and this strategy is more useful when a possible interaction set of an inferior is constructed and passed to its superior than vice-versa.

This coordination with abstraction is best suited to tree-structured systems (hierarchies).

The development of the idea of abstraction shows that there are several ways to generate abstractions, from open-loop to closed-loop, depending on the degree to which the local coordination can be determined a priori.

Each commander uses an intermediate model (abstractions) created by combining his perception of the situation (local model) with the assessment of the current situations received. The local decision takes these abstractions into account to minimize the worst possible amount of interactions necessary to make a decision. Since the abstraction will show that some interactions are infeasible, the set of possible worst cases is reduced, and efficiency improves.
6.2.3. Other structures than hierarchies

The previous section dealt with the problem of coordinating the activities of an organization with a singly connected structure, where the objective was minimizing the amount of interactions incurred. The following section will examine the four strategies proposed for applicability to other kind of topologies.

A. Static strategies

The static strategy can be extended with little difficulty to doubly connected topologies and to acyclic topologies if the communication principles are relaxed somewhat. It will not work with general topologies as, for example, cycles preclude the use of an induction order $H$.

The extension to doubly connected topologies is made by introducing a closer approximation for the messages-to-produce or respond functions by restricting them on a particular response. For an doubly connected topology, an induction relation $H$ can always be found exactly as in the singly connected case.

Acyclic topologies pose a more difficult problem. If decision $D_1$ affects decision $D_4$ through both $D_2$ and $D_3$, the amount of information transfer does not, in general, equal the sum of the interactions between $v_1, v_2$ and $v_1, v_3$. Thus, the costs to commanders in such a structure depend in a non-separable way on the interactions with each of their neighbours. This fact, plus the observation that one cannot define an induction order on an acyclic structure, means that an acyclic structure cannot be coordinated using the static scheme. However, acyclic topologies can often be restructured, by combining cells, into a doubly or singly connected system.

B. Coordination with no communication

By virtue of the fact that the ‘no communication’ strategy completely decomposes the coordination scheme into searches for internal coordination and that decisions are made by each commander completely independent of all others, this strategy is applicable to all topologies.
Strategies in teams, and groups are reduced to act on an individualistic basis when losing all capacities to communicate. This is the strategy of the four strategies (A - D) that is applicable to the heighest number of different types of organization (see also Table 6-1).

C. Limited-horizon coordination

The results from the strategy for hierarchies can be applied equally well to limited-horizon coordination of other structures than hierarchies by restricting attention to T steps of the future and using uncertainty (in relation to no-communication) to model the remaining time.

D. Coordination with abstraction

The coordination and propagation of abstract models (possible interaction set) for other structures than hierarchies throughout an FA model poses some difficulties. But, flexibility is offered as possible interaction sets used by this strategy are separable (Figure 6-5) and can at least be constructed in a cyclic topology.
Figure 6-5: Separation of possible interaction sets

For acyclic systems in general the strategies A-D can be constructed. Inferior cells construct, separate, and communicate messages to those they affect. Other commanders repeat this process when they have received all appropriate messages. Finally, the superior cells receive messages and commence on-line coordination.

6.2.4. Summary

Table 6-1 summarizes the applicability of the coordination strategies to various types of topologies. Singly connected systems are suited to the greatest variety of strategies. However, hierarchies have an advantage when abstraction is used. Certain strategies are useful with either acyclic
or doubly connected topologies but only the one with no communication works well with both.

Note that technically for the finite horizon and abstraction strategies there must be an incentive for each commander to expend the effort to produce the required decision which benefits only those commanders to which it is passed. Also, for system structures which contain several groups, each acting as a team of individual commanders, the coordination strategies can be used within groups and 'individual' strategies can be used between groups.

<table>
<thead>
<tr>
<th>Strategy</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>System structure</td>
<td>Static</td>
<td>No Comm.</td>
<td>Finite horizon</td>
<td>Abstraction</td>
</tr>
<tr>
<td>Teams:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Singly connected</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Doubly connected</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Acyclic</td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>General</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>

Table 6-1: Summary of applicability of coordination strategies to different organizational structures.

In this section strategies for distributed decision-making in relation to different topologies of a military organization were discussed. These strategies are defined in PROLOG rules and become part of the modelbase.
Another part of the modelbase is an uniform algorithm that is provided for each cell of an organization. This algorithm handles a class of temporal reasoning problems that occur in each cell of such an organization. These problems involve events whose order is not completely known.

6.3. Algorithm for predicting consequences

Each commander can make predictions about the consequences of his decisions by using an algorithm that is provided to each cell. As various grades of communication between cells affect each commanders uncertainty about future events, prediction quality can be expected to improve as communication increases.

Fundamental variables at a particular cell that may influence the quality of predictions are the current information load and the timeliness of that information. The concept of load is used in an abstract way to measure efficiency, and represents a variety of different actual demands upon the communication capabilities of the DDSS (see previous section). Timeliness is critical, because it has direct influence on the degree of uncertainty under which decisions have to be made.

In a cell of a military organization events are interpreted as incoming messages, decisions made by other commanders, own decisions, or milestones when something important has happened. An algorithm that reports useful bounds for consequences of such events is provided. The complexity of predicting consequences is examined in relation to the Command and Control cycle.

Reasoning about the consequences of events is important in a wide variety of military applications. In military HQs, a commander is often forced to contend with incomplete knowledge about the events he or she is concerned with. In particular, it is often the case that the commander does not know the exact order in which events occur.

It may be that the order in which certain events come about is not important to a decision-maker’s objectives. For instance, an author may not
provide the reader with the exact time of all *events* mentioned in a narrative, knowing that it is not critical that the reader have that information to follow the story. In other cases, however, figuring out the order of *events* is an important part of the task (e.g. unraveling a mystery or planning the sequence of operations of an attack).

The basic reasoning task we are concerned with can be stated as follows: Starting with general knowledge about the cause-and-effect relationships of a given domain, and specific knowledge about a particular set of circumstances, infer what is true over certain intervals of time. The task involving *events* whose order is unknown is somewhat complicated because the order in which *events* occur may determine what is true at various times. Where partially ordered *events* are concerned, we are interested in whether a fact is true over an interval in some total orders consistent with the given partial order (see 2.1 for definition of partial ordered sets).

6.3.1. Problem presentation

A Decision Key Task (section 1.4) is decomposable into smaller sub-tasks and can be expressed in terms of logically interconnected PROLOG-statements (in the following text *statements* are always PROLOG-statements) about the domain (see Chapter 5). Some *statements* are inferred from others, or explained by them. There is one *statement* that does not explain any other *statement*, called the principle 'goal'. In other words, this is the decision that has to be taken. Every remaining *statement* explains at least one other *statement*. *Statements* that are not explained correspond to non-decomposable, directly solvable sub-tasks. We call such *statements* facts.

The following terminology to present a decision problem has also been used in Chapter 2 for logic programming.

*Statements* correspond to *events* (e.g. incoming messages, decisions made by other commanders, own decisions, etc.). We shall refer to some *statements* as variables. The principle goal and facts are not variables. A variable which is used to explain another variable is an antecedent, and a
variable explained is a consequent.

We denote by $E = \{e_0^1, e_1, \ldots, e_n\}$ the sequence of all goals, variables, and facts that occur in a decision problem $E$. $e_0$ is by convention the principal goal. Let us denote $\{0, \ldots, n\}$ as $I$. The index set of facts is denoted as $I_f$, the index set of goals as $I_g$, and the index set of variables as $I_e$.

The permissible values of a variable are taken from the set $\{\text{true}, \text{false}, \text{any}\}$. The value 'any' corresponds to the situation when a variable may either 'true' or 'false' without influencing the values of its consequent. We assume that the required value of the principal goal $e_0$ is 'true', and that there are at least two variables in $E$, so that $e_0$ is not a fact.

The value of a goal $e_i$, $i \in I_g$, depends on the values of variables that explain $e_i$. To stress our goal-directed approach, we express this dependence as a rule which intuitively says that $e_i$ is true if and only if some combination of $e_j$ is true.

Now, an instance of a decision problem can be specified by:

1. a set of event types, $T = \{\tau_1, \tau_2, \ldots, \tau_n\}$;
2. a set of conditions, $P = \{p_1, p_2, \ldots, p_m\}$;
3. a set of causal rules, $R = \{r_1, r_2, \ldots, r_k\}$, $r_i \in I_g$, of the form $\langle \tau, \phi, \alpha, \delta \rangle$ composed of:
   - a triggering event type $\tau \in T$,
   - a set of antecedent conditions, $\phi \subseteq P$,
   - a set of added conditions, $\alpha \subseteq P$,
   - a set of deleted conditions, $\delta \subseteq P$,
4. a set of actual events, $A$, where $\forall e \in A$, type($e$) $\in T$;
5. a set of initial conditions, $\chi \subseteq P$;
6. a partial order $\leq$ on $A$.

For a given partial order $\leq$, we can identify a set of totally ordered extensions of $\leq$, $\{s_1, s_2, \ldots, s_k\}$. Each totally ordered extension $s_i$ defines a sequence $e_{1}, e_{2}, \ldots, e_{|A|}$ where $e_j \in A$ for $1 \leq j \leq |A|$. For each totally ordered
extension \( s_i \) and event \( e \in A \), we define index \((e, i)\) to be \( j \) such that \( e_j = e \) in the sequence \( e_1, e_2, \ldots, e_{|A|} \) determined by \( s_i \).

Let Possible(\( e \)) correspond to those conditions that hold immediately following the event \( e \) in some totally ordered extension. Let Necessary(\( e \)) correspond to those conditions that hold immediately following the event \( e \) in every totally ordered extension.

The basic idea is to compute useful bounds on Necessary(\( e \)) and Possible(\( e \)), where an upper bound corresponds to a subset of Possible(\( e \)), and a lower bound corresponds to a superset of Necessary(\( e \)). It is easy to specify some upper and lower bounds on Necessary(\( e \)) and Possible(\( e \)), since

\[
\forall e, \emptyset \subseteq \text{Necessary}(e) \subseteq \text{Possible}(e) \subseteq P.
\]

It is a bit more difficult to compute useful bounds. The approach taken here is to compute those consequences of an event \( e \) that follow from the consequences of events that immediately precede \( e \), \( \text{Pre}(e) \). For each event \( e \), we define \( \text{Maybe}(e) \) to be the set of all \( p \in P \) such that either

\[
p \in \text{Pre}(e) \land \neg \left( \exists \langle t, \phi, \alpha, \delta \rangle \in R, \left( \text{type}(e) = t \right) \land \left( \phi \subseteq \text{Pre}(e) \right) \land (p \in \delta) \right)
\]

or

\[
\exists \langle t, \phi, \alpha, \delta \rangle \in R, \left( \text{type}(e) = t \right) \land \left( \phi \subseteq \text{Pre}(e) \right) \land (p \in \alpha).
\]

If \( \leq \) defines a total order on the events in \( E \), then

\[
\forall e, \text{Maybe}(e) = \text{Necessary}(e) = \text{Possible}(e).
\]

Hypothetical reasoning (that is, reasoning in which facts or conclusions must be retracted in light of new information), suggested here, refers to solution approaches for predicting consequences of events, in which assumptions may have to be made to enable a search procedure to proceed. However, later along the search path it may be found that certain assumptions are invalid and therefore have to be retracted. This non-monotonic reasoning can be handled in a variety ways.
One approach to make assumptions that reduce the difficulty of the computation is to maintain multiple solutions and to discard those which evidence contradicts. Another approach is to keep track of the assumptions that support the current search path and to backtrack to the appropriate branch point when the current path is invalidated. This latter approach has been referred to by names like non-chronological backtracking. A related capability is truth maintenance which removes derived beliefs when their conditions are no longer valid. The approach followed in this study is presented in the next section.

6.3.2. The prediction procedures

Our approach to model the assumptions that should enable a search procedure to proceed is meant as the choice from the currently available alternatives or the choices of the direction of search for new alternatives.

Consider the simple abstract decision problem (Example 1) in which \( P = \{F, P, Q, R, S, T\} \), \( T = \{1, 2, 3, 4\} \), \( \phi = \{F, P, Q, S, T\} \), and \( R \) consists of the following rules (for definitions see 63.1):

\[
\langle 1, \{F\}, \{T\}, \{F\} \rangle,
\langle 1, \{T\}, \{F\}, \{T\} \rangle,
\langle 1, \{P\}, \{Q\}, \emptyset \rangle,
\langle 2, \{P\}, \{S\}, \emptyset \rangle,
\langle 3, \{P, S\}, \emptyset, \{P\} \rangle,
\langle 4, \{Q\}, \{R\}, \emptyset \rangle,
\langle 4, \{Q\}, \{T\}, \{F\} \rangle.
\]

The configuration of events for this example is shown in Figure 6-6.
Figure 6-6: The configuration of events

Each event instance is drawn as a circle enclosing its type. Under each event instance is an index that allows us to distinguish between different event instances of the same type. To compute useful bounds on Necessary(e) and Possible(e), we have to consider not only the events that precede e, but also the events that are unordered with respect to e.

In the rest of this section we describe an algorithm that computes two sets: Strong(e) and Weak(e), for each e ∈ A. We show that Strong(e) provides an upper bound for Necessary(e), and Weak(e) provides a lower bound on Possible(e). To demonstrate the usefulness of the bounds, we provide examples comparing Strong(e) and Weak(e) to Necessary(e) and Possible(e).

The top-level procedure decido takes the set of events E corresponding to the set of actual events A. decido computes the consequences of an event e only after computing the consequences of all events preceding e. For each event e, decido finds the set of B of events immediately preceding (before) e, and the set U of events unordered with respect to e. The two sets, B and U, are used to compute the consequences of e.
Figure 6-7: Events immediately before and unordered with respect to e.

Procedure: \texttt{decido} (E)
begins
\begin{align*}
E' & \leftarrow \text{sort}(E, \leq); \\
\text{until} & \ E' = \{\} \\
& e \leftarrow \text{pop}(E'); \\
B & \leftarrow \{e' | (e' \leq e) \land \neg(\exists e'', e' \leq e'' \leq e)\}; \\
U & \leftarrow \{e' | (e' \preceq e) \land (e \preceq e')\}; \\
\text{help-decido} & (e, B, U); \\
\text{end} \ loop; \\
\text{end};
\end{align*}

In the course of computing the consequences of a given \textit{event} $e$, there is a need to distinguish between different instances of the same condition being added or deleted. We make such a distinction on the basis of \textit{events}
and conditions required to bring about a particular addition or deletion. A
token corresponds to a particular instance of a condition changing as a
consequence of some event. The tuple \( \langle p, e, J \rangle \) consists of the condition \( p \)
and the event \( e \) which adds or removes condition \( p \) as a consequence of
the underlying rules. \( J \) is the set of preconditions in the rules concerned.

In the process of computation the consequences of a particular event \( e \),
we use the special event \( \hat{e} \) in referring to tokens that correspond to the
consequences of events that precede \( e \). Tokens are used to perform the
bookkeeping required to compute \( \text{Strong}(e) \) and \( \text{Weak}(e) \) for a given
event \( e \). This bookkeeping is necessary to ensure that all possible
consequences of a set of unordered events are generated. If there are a
finite number of such consequences, this process terminates. We employ
the PROLOG convention of notating variables that are never referenced by ‘\( \_ \)’.

For ease of exposition, we describe the process of computing the
consequences of an event in terms of two procedures: \texttt{help-decido} and
\texttt{konsekvenco}. The procedure \texttt{konsekvenco} applies the rules in \( R \) and
generates possible consequences; \texttt{help-decido} analyses the possible
consequences and computes \( \text{Strong}(e) \) and \( \text{Weak}(e) \). The procedure
\texttt{help-decido} takes an event \( e \), a set \( B \) of events immediately preceding \( e \),
and a set \( U \) of events unordered with respect to \( e \). It is defined as follows.

\begin{verbatim}
Procedure: help-decido (e,B,U)
begin
  S \leftarrow U_{e' \in B} \text{Super-strong}(e');
  W \leftarrow \cap_{e' \in B} \text{Super-weak}(e');
  \langle T, T' \rangle \leftarrow \text{konsekvenco}(U,W);
  P \leftarrow \{ p \mid (\langle \varphi, e', \_ \rangle \in T) \land (e' \neq \varnothing) \};
  Q \leftarrow \{ p \mid (\langle \varphi, e', \varphi \rangle \in T') \land \text{nedepend}(e, \langle \varphi, e', \varphi \rangle, T) \};
  A \leftarrow \{ p \mid (\langle \varphi, e, \varphi \rangle \in T) \land (\varphi \subseteq S - Q) \};
  F \leftarrow \{ p \mid (\langle \varphi, e, \varphi \rangle \in T') \land (\varphi \subseteq S - Q) \};
  Q^+ \leftarrow Q \cup \{ p \mid (\langle \varphi, e, \_ \rangle \in T') \};
  \text{Strong}(e) \leftarrow (S - Q^+) \cup A;
  \text{Weak}(e) \leftarrow \{ p \mid (\varphi, \_, \_ \rangle \in T \} - F;
\end{verbatim}

93
\[
\text{Super-strong}(e) \leftarrow \text{Strong}(e) - \{p|\langle \varphi, \_\_\rangle \in T'\}; \\
\text{Super-weak}(e) \leftarrow \text{Weak}(e) \cup (F \cap P);
\]
end;

In proving that \text{decido} works as advertised, we have to delve into the details of \text{help-decido}. First, it will be necessary to understand \text{konsekvencoj}. The only thing we need from \text{konsekvencoj} is an understanding of \(W\). The function \text{nedependa} will be defined and explained later. The set \text{Strong}(e), as computed in \text{help-decido}, is meant to include only conditions that hold following \(e\) in all totally ordered extensions. The set \text{Weak}(e) is meant to include all conditions that might hold following \(e\) in some totally ordered extension. In addition to \text{Strong}(e) and \text{Weak}(e), \text{help-decido} also computes the sets \text{Super-strong}(e) and \text{Super-weak}(e) used in defining \(S\) and \(W\) for \textit{events} that immediately follow \(e\). \(S\) is meant to include only conditions that must hold immediately preceding \(e\) in all totally ordered extensions, ignoring the consequences of the \textit{events} in \(U\). We note that

\[
W \leftarrow \cap \{e|e \in B \text{ Weak}(e')\}
\]

would exclude conditions that we require, whereas

\[
S \leftarrow \cup \{e|e \in B \text{ Strong}(e')\}
\]

would include conditions that we do not require. The difference between \text{Strong}(e) and \text{Super-strong}(e) is that the latter excludes conditions that might be deleted following \(e\) in some totally ordered extension by an \textit{event} currently unordered with respect to \(e\). Similarly, \text{Super-weak}(e) includes all of \text{Weak}(e) plus those conditions that might be added following \(e\) in some totally ordered extension by an \textit{event} currently unordered with respect to \(e\).

Assuming that \text{Super-strong}(e) and \text{Super-weak}(e) are as indicated, it should be clear that \(S\) and \(W\), as computed in \text{help-decido}, contain what we require. To explain the use of intersection in defining \(W\), note that \text{Super-weak}(e) excludes conditions that will be deleted as a consequence of \(e\) in all totally ordered extensions and not added as a consequence of any \textit{event} unordered with respect to \(e\). For each \(e'\) immediately preceding \(e\), \text{Super-weak}(e') excludes only those conditions that \(e'\) is directly
responsible for deleting. In computing \( W \), we need to take the intersection to exclude all such conditions. An analogous argument can be used to explain the use of union in defining \( S \).

The procedure \texttt{konsekvencoj} computes the sets \( T \) and \( T' \) of tokens corresponding to all those conditions that might be, respectively, added or deleted by \textit{events} in \( U \). It is defined as follows:

Procedure: \texttt{konsekvencoj}(E,P)

begin
\( T \leftarrow \{ \langle p, \delta, \epsilon \rangle \mid p \in P \} \);
\( T' \leftarrow \{ \} \);
for \( t \in T \), \texttt{support}(t) \leftarrow \{ \} \;
flag \leftarrow \text{true} \;
while flag
\( \text{flag} \leftarrow \text{false} \;
\text{for } \langle p, \_,-, \_ \rangle \in T \;
\text{for } e \in E \;
\text{for } \langle r, \phi, \alpha, \delta \rangle \in R \text{ such that } (\text{type}(e) = T) \land ( p \in \phi) \;
S \leftarrow \texttt{trovu-subteno}(e, \phi, T);\)
\text{if } S \neq \text{nil} \text{ then}
\text{for } a \in \alpha \;
\text{if } \text{support}(\langle a, e, \phi \rangle) \neq S \text{ then}
\text{flag} \leftarrow \text{true} \;
support(\langle a, e, \phi \rangle) \leftarrow S \;
T \leftarrow \{ \langle a, e, \phi \rangle \} \cup T \;
\text{end if};\)
\text{end loop};
\text{for } c \in \delta, T' \leftarrow \{ \langle c, e, \phi \rangle \} \cup T';\)
\text{end if};\)
\text{end loop};
\text{end loop};
\text{end loop};
\text{end loop};
\text{return } \langle T, T' \rangle;\)
end;
Chapter 6

The procedure **konsekvenco** takes a set of *events* E, and a set of conditions P. By the operation of **help-decido**, P includes all of those conditions that might hold preceding some *events* in E in some totally ordered extension. It also ignores the consequences of those *events* in E. **konsekvenco** starts with a set of tokens corresponding to the conditions in P, and, using the rules in R, generates additional tokens using the *events* in E as triggering *events*. Before we describe in detail the operation of **konsekvenco**, we will define the operation of the one subroutine that **konsekvenco** relies upon.

The procedure **trovu-subteno** takes an *event* e, a set of conditions P, and a set of tokens T, and returns a set of tokens. Each set of tokens constitutes one instance of support for the token, with one token of the appropriate type of each condition in P. **trovu-subteno** performs the bookkeeping necessary to ensure that **konsekvenco** generates neither duplicate nor causally impossible tokens. It is defined as follows:

Procedure: **trovu-subteno** (e, P, T)

begin
if P = {} then
    return {{}}
else
    R ← {};
p ← first (P);
for S ∈ **trovu-subteno** (e, rest (P), T)
    for ⟨p, e', φ⟩ ∈ T
        if (e ∧ e') ∧ nedepend (e, ⟨p, e', φ⟩, T) then
          R ← {S U {⟨p, e', φ⟩}} U R;
        end if;
    end loop;
end loop;
end if;
end;
Chapter 6

The function \texttt{nedepend}a is designed to eliminate circularities in the support for any token added to $T$ or $T'$ by \texttt{konsekvencoj}. This is accomplished using a marking algorithm that is basically a depth-first search of a graph.

Procedure: \texttt{nedepend}a($e,t,T$)
begin
  for $t' \in T$
    if $t' = \langle \_, e, \_ \rangle$ then
      mark($t'$) $\leftarrow$ false;
    else
      mark($t'$) $\leftarrow$ unknown;
    end if;
  end loop;
return \texttt{help-nedepend}a($t$);
end;

Procedure: \texttt{help-nedepend}a($t$)
begin
  for $l \in \text{support}(t)$
    flag $\leftarrow$ true;
    for $t' \in l$
      if mark($t'$) = unknown then
        mark($t'$) $\leftarrow$ in-progress;
        mark($t'$) $\leftarrow$ \texttt{help-nedepend}a($t'$);
      end if;
      flag $\leftarrow$ mark($t'$) \land flag;
    end loop;
    if flag then
      return true;
    end if;
  end loop;
return false;
end;
Chapter 6

Now we can describe what it is that *konsekvencoj* computes.

<table>
<thead>
<tr>
<th>Index</th>
<th>Type</th>
<th>Strong</th>
<th>Necessary</th>
<th>Possible</th>
<th>Weak</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>{F, P, Q}</td>
<td>{F, P, Q}</td>
<td>{F, P, Q}</td>
<td>{F, P, Q}</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>{P, Q, T}</td>
<td>{P, Q, T}</td>
<td>{P, Q, T}</td>
<td>{P, Q, T}</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>{P, Q, S}</td>
<td>{P, Q, S}</td>
<td>{F, P, Q, S, T}</td>
<td>{F, P, Q, S, T}</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>{Q}</td>
<td>{Q}</td>
<td>{F, P, Q, S, T}</td>
<td>{F, P, Q, S, T}</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>{Q}</td>
<td>{Q}</td>
<td>{F, P, Q, S, T}</td>
<td>{F, P, Q, S, T}</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>{Q, S}</td>
<td>{Q, S}</td>
<td>{F, Q, S, T}</td>
<td>{F, Q, S, T}</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>{Q, S}</td>
<td>{Q, S}</td>
<td>{F, Q, S, T}</td>
<td>{F, Q, S, T}</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>{Q, S}</td>
<td>{Q, S, T}</td>
<td>{Q, S, T}</td>
<td>{F, Q, S, T}</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>{Q, R, S, T}</td>
<td>{Q, R, S, T}</td>
<td>{Q, R, S, T}</td>
<td>{Q, R, S, T}</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>{F, Q, R, S}</td>
<td>{F, Q, R, S}</td>
<td>{F, Q, R, S}</td>
<td>{F, Q, R, S}</td>
</tr>
</tbody>
</table>

Table 6-2: The results computed by decido for Example 1.

The purpose of this section was to analyze the problem of reasoning about partially ordered *events* and to provide an algorithm that computes useful bounds for the set of consequences of *events*. Our procedure *decido* fares good in Example 1, as it is indicated by the results shown in Table 6-2. It should be pointed out that, while toggle events are troublesome for our algorithm, there are many special cases that can be handled efficiently.

To separate out the issues central to distributed decision support with incomplete knowledge about the order of *events*, we chose Decision Key Tasks for specifying cause-and-effect relationships. An example is given in the next section.

### 6.4. A model of a military example

Military decision-making is a process where each participant influences the course of action and therefore influences decisions made by others.
The problem in military decision-making can be stated as follows: starting with general knowledge about the cause-and-effect relationships in, for example, crisis management, and specific knowledge about a particular set of circumstances (e.g. goals, events), the decision-maker must infer what is true over certain intervals of time.

With the example of the decision problem of 'declaring the status of alert' from the NATO 'Alert System', we illustrate the cause-and-effect relationships, apply the algorithms introduced in the previous section and discuss possible extensions. The requirements analysis (see Chapter 5) for the appropriate Decision Key Tasks in the 'Alert System', leads to the structure shown in Figure 6-8.

The example shows that the mission of a headquarters includes the promulgation and carrying out of a commander's decision. In general terms, the cells of a headquarters support a commander in decision-making by:

- Maintaining a factual information database of friendly and other forces
- Identifying and assessing the threat facing a headquarters
- Recommending courses of action for decisions by a commander
- Promulgating a Commander's decision
- Monitoring, assessing and documenting the results.
Figure 6-8: Key Task Structure for the 'Alert System'
Chapter 6

The prediction algorithms introduced in the previous section, for example, analyse a particular decision in relation to the Decision Key Task 'decide need for declaration of alert' to predict the consequences of such a decision.

A decision in relation to 'status of alert' may have a bunch of operational consequences. These consequences (called measurements in the execution phase) are taken from predefined lists and are checked against the results provided from previous phases. For example, relocation of forces are checked against logistic constraints.

It is unrealistic to assume that all operational consequences will be fully determined during the search for solutions in the set of possible consequences. Typically, some future event values will already been known, and an acceptable prediction will have to take these values in account. The prediction algorithm actually functions with regard to the proportion of events known before and events whose values can be determined by the algorithm. Three cases are possible:

**Case 1.** All events have known consequences. The algorithm uses forward reasoning to infer the value of the principle goal $e_0$. If its valuation is 'true', the given consequences already form a solution. A modification of this situation is necessary when not all consequences are known.

**Case 2.** No future events are known and we want to determine their values to get $e_0 = \text{true}$. In this situation the algorithm uses backward reasoning: assuming $e_0 = \text{true}$ the algorithm determines the required values of $e_0$'s antecedents, antecedents of these antecedents, until the values of facts are determined.

**Case 3.** Some events are known, but not enough to infer the value of the principle goal. In this case the algorithm is applied to the known events and continued until the value of some goal cannot be inferred because of the lack of an event value. The goals whose values have been found by the algorithm are treated as if they were known events, and are processed as in Case 2.
Known future events may be taken from the environment, or may be the result of the user's preferences. In the simplest solution, the user restricts some of the events to be 'true' or 'false' regardless of the structure of the problem. Then he wants to check whether these restrictions allow a solution.

One of the possible extensions of the algorithm may be the introduction of rules that describe an assessment of an enemy situation. A complete picture of an enemy's situation is usually not available, especially in terms of how the enemy himself sees his situation. However, partial representations can be derived as estimates of an enemy's likely view of elements of his situation.

During a crisis or war, the validity of rules may be verified, or rules may be changed, or abandoned.

The algorithm then should work as follows: If an own situation has been established on the basis of own rules, the rules for determination of the enemy response become operative. Predictions regarding the enemy response are possible to some extent through forward- and backward-chaining using non-monotonic logic on the basis of our assumptions in relation to the enemy situation. This improves the own situation from the point of view of more exact assessment of the effects of our measures.
7. HUMAN-FACTOR CONCEPTS AND SYSTEMS ARCHITECTURE

There is often a deep misunderstanding of the role of people in person-machine systems (Nagel [44]). Nagel points out that there is a belief in the possibility of engineering the human side of the operation in the way that one may hope to engineer the mechanical or electronic side. As the long list of human-factor failures in technical systems suggests, the attempts to implement this belief are often needlessly clumsy (Tolcott [74], Rose [75]). The extensive body of human factors research is often invoked at such a late stage in the design process that it can amount to little more than the development of warning labels and training programs for coping with inhuman systems. Common concomitants of insensitive design are situations in which the designers (or those who manage them) have radically different personal experiences from the users.

However, even when the engineering of people is sensitive, its ambitions are often misconceived. The complexity of systems places some limits on their correctness, making it hard to understand the intricacies of a design. As a result, one can neither anticipate all problems nor confidently treat those one can anticipate, without the fear that corrections made in one domain will create new problems in another.

Part of the genius of people is their ability to see (and hence respond to) situations in unique (and hence unpredictable) ways. Although this creativity can be seen in even the most structured psychomotor tasks, it is central and inescapable in any interesting distributed decision-making system. Once people have to do any real thinking, the system becomes complex. In such cases, the task of engineering is to help the users to understand the system, rather than to manage them as part of it.

A common sign of insensitivity in this regard is use of the term user error to describe problems arising from the interaction of user and system. A sign of sensitivity is incorporating users in the design process.

Given that human decision-making is the common denominator of DSS's, it seems appropriate to use it as a basis for constructing a distributed
Chapter 7

decision support system.

What is needed is an approach that applies cognitive engineering results to the development of a DDSS such that a basis for cooperative human-computer decision-making is established. A rule of thumb is that human problems seldom have purely technical solutions, while technical solutions typically create human problems.

Pursuing this line of inquiry can point to specific problems arising in distributed decision-making systems, and focus technical efforts on solving them. Hence a purely technologically driven development of a new architecture can produce unintended and unforeseen negative consequences, and should be avoided.

In our approach to configure a DDSS, it is assumed that the decision context can be defined, and a decision support system appropriate to that context can be built. The DDSS should be the product of that development process, and once implemented will play an invariant role in the decision process.

We suggest that application of the approach should begin one level higher than has often been the case, i.e. at the level of the decision system. A decision system is comprised of a human decision-maker (DM) and a computerized decision support system (DSS), and has as its purpose the production of decisions (section 73). Adopting the decision system perspective highlights the complementary roles of DM and DSS. It raises several issues not generally considered explicitly in the design of DSS: selection of a problem formulation, selection of a decision strategy, and allocation of tasks to the DM and DSS.

The following sections address the configuration of a DDSS's architecture and define the potential roles of DDSS, and, hence the scope that must be considered in its architecture. Possible levels of automation for manual and machine functions are discussed. Requirements for a dialogue between the human and the computer are presented.
7.1. Human-computer interaction

The objective of a human-computer interaction analysis is to define requirements in relation to an effective and efficient interaction between the decision-maker and the DDSS. As Seifert and Neujahr [76] point out, the spectrum of interaction between the human and the computer extends from fully manual operation to fully automated operations. They identify six automation levels for man-machine interface functions, which are widely accepted in the aircraft industry, and vary with respect to the share of authority assigned to the human (Table 7-1).

The fully manual and the fully automatic modes of operation need little explanation. Some essential human factors must be mentioned however: Manual operation often leads to operator stress, high workload, and fatigue. But too much automation is likely to lead to boredom, complacency, and erosion of competence. It may be dangerous in critical situations to move the commander too far out of the control loop by supplying too much automation.

<table>
<thead>
<tr>
<th>mode of operation</th>
<th>authority</th>
</tr>
</thead>
<tbody>
<tr>
<td>Human</td>
<td></td>
</tr>
<tr>
<td>Manual</td>
<td>hhhhhhhhhhhhhh</td>
</tr>
<tr>
<td>System-supported manual</td>
<td>hhhhhhhhhhhhh</td>
</tr>
<tr>
<td>Augmented, system-limited manual</td>
<td>hhhhhhhhhccc</td>
</tr>
<tr>
<td>Non-manual, but with unlimited manual supervision</td>
<td>hhhhhhhccc</td>
</tr>
<tr>
<td>Non-manual, but with limited manual supervision</td>
<td>hhhhhhhcccccc</td>
</tr>
<tr>
<td>Non-manual</td>
<td>ccccccceccccccc</td>
</tr>
</tbody>
</table>

Table 7-1: Automation levels for man-machine interface functions.
Some additional remarks must be made regarding the intermediate modes of automation. In the system-supported manual mode, manual control functions are augmented by automatic control systems as, for example, nose-wheel steering and stabilizers in aircraft. Decisions in this mode may be supported by electronic checklists, spreadsheets, integrated displays, or reversible functions for error correction. The authority remains in these cases entirely with the commander.

Examples for the augmented mode are data entry formatting and validation checks including autonomous input error correction. In the first non-manual mode, manual override of automatic functions is made possible.

The next mode is characterized by automatic functions with manual accept/reject capabilities. Typical examples are the approval of automatic target identification or prioritization, and the acknowledgement of a trimming computer’s proposal in a submarine. The degree of human authority depends on whether manual accept/reject is optional or mandatory, and hence may cover a wide spectrum. Sometimes manual sanction, while technically possible, is not practical due to time restrictions, as is often the case in target identification or prioritization tasks.

To identify the relative merits of the degrees of automation outlined above, let us assume that the decision support is provided by a human advisor. This line of thinking leads us to look in some more detail into the methods of information exchange among humans. Nickerson [77] has completed a comprehensive list of conversational attributes (Table 7-2).
Shared knowledge: Situational context  
Common world knowledge  
Special knowledge  
History  

Accepted conventions: Bidirectionality  
Mixed initiative  
Apparentness of who is in control  
Rules for transfer of control  
Sense for presence  
Structure  
Characteristic time scale  
Intolerance of silence  

Other attributes: Nonverbal communication components  
Wide bandwidth  
Informal language  
Peer status of participants  

Table 7.2: Some characteristics of human conversation

From this table the crucial role of shared knowledge and accepted conventions becomes obvious. The observation that peer status is required to ensure a serious conversation is of particular relevance in the context of decision support. DDSS-based advice will only be accepted if it is perceived as being of excellent quality.

7.2. Concepts

The general approach to DDSS development as suggested in Chapter 4 focuses on the rapid development of a DDSS architecture, a user interface and dialog for the DDSS. Through a process of rapid iterations, a carefully selected DDSS user group working with that architecture helps refine it and eventually defines 'the DDSS'.

However, beginning the analysis and architecture design of a DDSS in relation to a decision system forces a developer of a system architecture to
consider explicitly the capabilities of a DDSS and the DM and the alternative allocations of Key Tasks to these two decision system resources. This perspective recognizes the importance of the decision process in DDSS architecture, and the importance of selecting a decision strategy.

This selection and the allocation of decisions are interdependent, and can be taken dynamically: i.e., strategy and Key Tasks allocation need to be fixed when the DDSS is built, but can change to fit the situation, including the capabilities of the available resources.

A human decision-maker who works with a computerized DDSS to produce decisions can be modeled as a decision system. We assume that a decision system has the potential to produce more efficient decisions than those made by decision-makers without system support. The increase in efficiency\(^7\) and effectiveness\(^8\) of decisions is the result of an improved decision-making process. The DSS literature has often neglected the decision-maker's behaviour in reaching decisions and has implicitly assumed that it is outside the scope of DSS design. However, when we take the comprehensive view of the decision-maker and the DDSS as a decision system, we cannot ignore this stage (see Bots, [6]).

Descriptive studies of non-trivial decisions have shown (Michel & Teidel [78]) how decision-makers first select a decision strategy before they proceed to solve a problem. Moreover, a number of models have been proposed to describe the way decision-makers select decision strategies (e.g. Payne [79] and Bots [6]). In all these models, the perceived complexity of the problem plays a crucial role in selecting the decision strategy, as the DM attempts to match the complexity of the decision strategy to the demands of the problem.

In the design of a decision system, we also have to address this issue of how a DM selects a decision strategy. We will treat this issue in the context of three steps:

---

7. Efficiency: Taking action with the least expenditure of resources.
Chapter 7

- Development of a problem presentation,

- Selection of a decision strategy to solve the problem as formulated,

- Choice of human or computer resources to support the decision strategy.

The first step will be addressed in Chapter 8 by the STAR Table format. The second and third step will be elaborated below.

This selection of a decision strategy may also be considered as being analogous to the first stage of decision-making (intelligence) defined by Simon [33] (Chapter 3). The remaining stages of the decision process (design, choice, implementation) proceed after the need for a decision has been recognized and a decision about how to decide has been made.

A decision system has two types of internal procedures, those that effect its environment and those for managing its internal functioning. For example, in a decision system for the command and control cycle, the decision system has procedures that:

1) ensure the timely delivery of a high-quality situation report, and
2) manage the message flow within the command and control cycle.

The decision system (DS) has a 'meta-system', the organization in which it functions, e.g. a military headquarters. The meta-system determines the goal of the decision system and affects its purpose (e.g., pushing for more timeliness of reports). The organization is therefore the decision system’s client.

The decision system utilizes two resources to accomplish its tasks: the human decision-maker (e.g., a commander), and the computerized DDSS. Notice that decision-makers may have several roles in the decision system formulation, e.g. they may decide about how to allocate resources in order to complete a Decision Key Task and also be the decision system’s client. We shall, therefore, refer explicitly to these distinct roles. Furthermore, we shall refer to only one DM without loss of generality, though there could
in fact be several decision-makers involved.

The DS configuration is pictured in Figure 7-1 and Figure 7-2, which have been adopted and refined from (Te'eni [80] and Ginzberg [81]).

Figure 7-1: Decision system (DS) configuration

7.3. Functional DDSS design

The DS can be functionally decomposed into three subsystems, each of which performs one major function. The subsystem that controls the DS decides on a particular decision strategy and the human-computer interaction necessary for its implementation. The distributed decision-processing subsystem executes the decision strategy using the resources assigned to it. And the human-computer interaction subsystem provides the appropriate communication channel between the commander and the other two subsystems to make the DS functional.
Figure 7-2: Basis decision support system configuration

The DS control subsystem manages the DS functioning. Specifically, this subsystem recognizes the need for a decision, decides on a decision strategy, allocates Decision Key Tasks to the DS resources, and decides on the appropriate human-computer interaction for the selected decision strategy and task allocation.

The distributed decision-processing subsystem executes Decision Key Tasks that are directed to it by the control subsystem, and in addition, the
distributed decision-processing subsystem manages its own internal functioning. In our Command Control example, the distributed decision-processing subsystem might forecast consequences of events, and the decision-strategy execution function involves all phases of the command and control cycle, from maintain status to order execution.

The human-computer interaction subsystem handles a pattern of human-computer interactions that has been determined by the control subsystem as appropriate for a specific combination of a decision strategy and a Key Task allocation. The objective of the human-computer subsystem is to ensure an effective and efficient interaction between the decision-maker and the DDSS to facilitate DS tasks. For example, the DDSS (functioning as a human-computer interaction subsystem) may validate input and alert the user to any input errors.

In a future DS, the human-computer interaction subsystem might deal with complex issues such as assuring a shared understanding between the DM and the DDSS; e.g., letting the DM know whether the DDSS has correctly understood the problem and is doing what the DM thinks it is doing, or informing the DM whether the DDSS is performing the functions that the DM is currently not attending to.

Our understanding of the decision process and the DS functions and resources must be put in the context of distributed decision-making that adapts to varying conditions. It follows that the distributed system control should be designed in the light of the actual contingencies. For example, DSSs should assist the DM in detecting the appropriate decision style for specific conditions.

The decision system configuration described here provides a perspective on DDSS and DDSS architecture that is concerned with human behaviour because this configuration causes us to focus on the decision process and to see the DDSS as a resource to the system that carries out that process. It clarifies the relationship of DDSS and decision-makers as complementary resources to the decision process. The allocation of Decision Key Tasks between DDSS and DM must be given explicit consideration in relation to the different levels of automation (section 7.1). In summary, the decision
system perspective forces the designer to conceptualize a DDSS that recognizes the need for a decision, selects a decision strategy in response to that need, and manages the implementation of that strategy.

Now that we have a clearer view of what functions are needed in the DS, we can formulate a way in which a DDSS architecture can be defined.

7.4. System architecture

The familiar supervisory Command and Control system architecture described by Sheridan and Hennessy [82] already contains special system functions for commander support. The essential point is that a distinction is made between human and task-interactive processors. A possible architecture for a distributed decision support system using these special processors is presented in Figure 7-3. It is based on proposals made by Rouse [75]. As may be seen from this figure, a set of three different databases has been introduced into the system as a complement to the human interactive processor.

Stored in these databases are PROLOG rules about the state of the work domain, the system state, and the human operator's state and goals. First, to identify user state and goals, a collection of active goals (Key Tasks), plans, and scripts is needed (compare Chapter 5). Secondly, knowledge about system functioning must be encoded in specific rules (strategies; Chapter 6) and procedures (prediction of consequences; Chapter 6). Lastly, environmental conditions and operational phases (see Command and Control cycle; Chapter 2) describe the state of the work domain (cell state; Chapter 4). By making use of this knowledge, the 'human interactive processor' may produce intelligent and adaptive behaviour at the user interface. Adaptivity of this interface may be desirable with respect to environmental conditions, operational phases, system states, or user performance characteristics. Main functions of the human interactive processor are interface management and distributed decision processing. Both receive input from a database containing user information.

Dialogue design is a particularly essential aspect of interface management. The selection of a suitable dialogue form is an efficient means to match
the method of data exchange to the needs of the user. An example of the
dialogue form is described in detail in the next Chapter.

Decision-processing functions may take the form of predicting the
consequences of events as discussed in Chapter 6. A different class of
decision-processing functions in a distributed system (not considered for
the prototype described in Chapter 8) may be devoted to error
management. The goal is to build error-tolerant systems by the automated
identification, compensation, and prevention of errors. As a simple
example, typing errors may be considered, which often occur because not
the intended key but a neighbor is activated. Identification and correction
of a typing error can, in a first step, be achieved by making reference to a
thesaurus. Remaining ambiguities can be further resolved by analyzing the
local neighbours of an error letter on the keyboard, and use the result for
the automatic compensation of misspellings.

A DDSS that embodies an expectancy of the kinds of errors that might
occur may also be able to detect errors automatically as deviations from
normative procedures. Systems with such capabilities are said to be
good-sharing and intent-driven, Wiener [83].

The identification of errors may again be based on the databases
mentioned above. Using this information, errors of omission or
commission may be identified and compensated for by making reference
to stored legal procedures. This class of decision-processing functions has
not attracted attention yet, and would be a good subject for further
research.
Figure 7-3: Distributed decision support system architecture
Figure 7-4 provides a more detailed description of the DDSS-LAN configuration shown in Figure 7-3. This LAN implementation has the capability to extend its application beyond LANs to wide area networks as well (please refer to Chapter 8).

LAN: Local Area Network
ACC: Access Control Centre
TNUI: Trusted Network Interface Units

Figure 7-4: DDSS-LAN architecture

As mentioned several times in this study, it is very essential that DDSS development considers results from human factors research. Important features are the cognitive processes of decision-making and the mental task of conceptualization on the part of the DDSS's user. This is not a
trivial task because the spectrum of commander behaviour, for example, is rather broad. Commanders may, for example, be naive or expert, either in computer handling or in the subject matter, or in both. In addition, their use of the DDSS may be casual or frequent (Figure 7-5). Thus, one commander may show skilled behaviour, and another rule-based behaviour, in identical tasks. An architecture of a DDSS intended to incorporate human behaviour has therefore to cover these relevant user dimensions.

![Diagram showing user dimensions](image)

**Figure 7-5: Some relevant user dimensions**

The actual level of action needed to perform a particular Key Task depends on its difficulty and on the available level of training and expertise. Consequently, the information and decision support needs cannot be identical for the whole spectrum of possible users. While, for example, a casual novice will need tutoring in the subject matter as well as in computer handling, the frequent expert user needs high-level advice in the subject matter. Also, a skilled commander will after an extended break resort for a while to rule-based or knowledge-based behaviour until a reasonable level of training is reached again.

Therefore, it is an essential design goal for DDSS to give the user as complete a picture of what is going on in the system as possible. One
obvious way to go is the substitution of textual information by graphical displays, which can be read and understood much faster.
Chapter 8

8. EXAMPLE OF A DDSS

This Chapter describes DECSUP, a prototype DDSS whose development is based on this study, and its implementation. DECSUP is designed to support decision-makers in military headquarters. DECSUP has been set up for use in a UNIX environment. To date, this system has been used extensively to test and analyze strategies and procedures for distributed decision-making.

Since the vulnerability of DECSUP is of major concern to a military headquarters and influences the implementation, discussions on communications security and data integrity are included in this Chapter.

8.1. Preconditions

For the project at hand it was not possible to identify deficiencies and requirements for distributed decision-making solely by extraction from approved military documents as suggested in Chapter 5, because there was only one study investigating a headquarters in relation to Decision Key Tasks as an integrated entity (Schlickmann [2]). This study shows results that are usable in relation to only some of the Decision Key Tasks, because the concept of Key Tasks is not completely understood by all military users.

At the headquarters we were concerned with are cells who have pursued and are pursuing partial Command and Control systems aspects under defined objectives of other NATO projects. Whilst those projects have generated approved deficiencies and requirements, they do not have a responsibility to consider Key Tasks in relation to a DDSS.

With this lack of input it was necessary to consult a variety of sources: users at a command post exercise, C2 systems managers, and documentation of current projects. But first and foremost it was necessary to apply common-sense judgment by comparing current systems capabilities with the putative working method of a headquarters.
Specific deficiencies of information systems in military headquarters identified by a 'headquarters information systems integration study group' included among others:

"A lack of decision support bringing together all sources of Command and Control information." This precludes the possibility of presenting an automated, all-encompassing view of the military situation.

Operational requirements were identified by the study group mentioned above to rectify deficiencies in headquarters Command and Control information systems. The operational requirements of decision support included among others:

"Provide a Decision Support System in the headquarters' cells to allow the Command Group (Commanders in a headquarters) to access data, files, reports, and databases residing on various systems."

From these broad operational requirements, specific functional requirements were determined for each component of a headquarters. These functional requirements for decision support capabilities included algorithms like:

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Supported decision making aspect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process modelling</td>
<td>The use of models to predict the outcomes of the current situation given proposed courses of action</td>
</tr>
<tr>
<td>Value modelling</td>
<td>The quantitative representation of subjective preferences among possible courses of action, to allow precise differences in the desirability of these possible actions to be identified</td>
</tr>
</tbody>
</table>
Machine learning
Compensation of systematic inconsistencies or biases for judgments refining and amplifying

Adaptive aiding facilities

Automated reasoning
Numerical techniques
Calculation of conditioned probabilities and utilities. Search for success paths in problem spaces.

Symbolic techniques
Generation and validation of hypothesis by rule-based forward and backward inferencing.

These capabilities were defined by the above-mentioned working group after reviewing existing decision 'aids' and analyzing them to determine the capabilities and characteristics of current decision-aiding technology. To the surprise of the working group, existing complete 'decision aids' were found to be highly specialized and not directly applicable to any of the identified decision situations described by Decision Key Tasks. In fact, these 'decision aids' were found to be applicable only to the (oftentimes artificial) problems for which they were built.

8.2. Software design and functionality

This section will discuss the software structure of DECSUP as well as its functionality, and a tool for rule-base verifications followed by a discussion of the user interface for DECSUP.

8.2.1. Introduction

A distributed decision support system that is part of the NATO-CCIS shown in Figure 8-2, is one that is typically viewed as 'difficult to program.' Difficulty arises from two sources:
Chapter 8

1) Message-based interprocess communication between cells;
2) Asynchronous, non-deterministic ordering of computations.

However, before deciding as to the inherent difficulty of designing efficient software for DECSUP, available knowledge must be employed towards understanding the software characteristics.

Starting with the requirements analysis in relation to Decision Key Tasks (see Chapter 5) and adding information from other sources as stated in 8.1, Chapter 7 provided a model that accurately characterizes the structure and semantics of the Key Tasks introduced in Chapter 1.

This is because a model of the method of working in a military headquarters is given by the Key Task relationships. Since a headquarters consists of cells, and each cell performs at least one Key Task, there is a mapping between the structure of a headquarters and the Key Task relationships. The general characteristics of the necessary software in DECSUP used in each cell of a military headquarters to perform a Decision Key Task are therefore:

- Distributed points (cells) of decision-making authority.
- Globally defined (but locally achieved) efficiency objectives.

In the rest of this Chapter, structure and employment of the software comprising DECSUP are presented, the user interface is delineated, and some implementation aspects are discussed.

8.2.2. Software structure

The development of software for the solution of a certain problem is a function of many factors. In the case of DECSUP, modular design of the software should allow both easy reconfiguration by users and also rapid implementation of the software itself (less than four man-months). From the requirements posed to DECSUP (see section 8.1) we can deduct that it must at least
- comprise the ability to retrieve data from databases held in other systems;

- handle incoming and outgoing messages;

- handle status information and transition functions to perform certain strategies;

- provide reasoning algorithms for predicting consequences of events.

We followed the prototype approach introduced in Chapter 4. This implies the following interaction of the modules of DECSUP at the highest level (Figure 8-1). A detailed description of each module is provided later in this Chapter.
Some of the modules are static source-level implementations of the model introduced in Chapter 6 (e.g., scenario, reasoning algorithm), which do not vary. Some modules represent task-dependent definitions of algorithms (contents of internal state vector and FA definition), efficiency objectives, communication topologies, and system workloads. Most of the task-dependent components have simple default specifications that allow rapid configuration of a test module for simple tasks. What follows is a
brief discussion of the function and design of the modules shown in Figure 8-1.

8.2.3. Software functionality

**Scenario:** This module, written in about 300 Prolog rules, embodies the driving component of DECSUP for testing purposes. It contains the definition of Key Tasks and in fact processes events and maintains virtual time. Standard random variable generators are available for creation of events according to the user's specification. Separate event lists are maintained for each cell of the structure. Virtual time is defined by the way in which users provide actions to take place when an event occurs. In this module, virtual time is represented by discrete message-passing intervals. Finally, there is a function that is called to update evaluation related information. This task-dependent function allows DECSUP to be flexible with respect to evaluating algorithms which may have widely varied objectives.

**FA definition:** The components of the finite automata (FA) module of DECSUP are specified in PROLOG and consist of three logical parts (see also Chapter 6). The first is a PROLOG structure that specifies the components of the FA. The second logical definition that must be made is the initial state $s_0$ for each cell $v_j$. Finally, the transition functions (from one cell status to another; see also 4.4.3) must be defined. This is accomplished by specifying the actions of each transition allowed in the definition of the distributed FA upon the headquarter's cells.

The remaining elements of Figure 8-2 represent the action of DUELING and the output of the PROLOG interpreter. The elements before the PROLOG interpreter are specified in PROLOG, and hence permit the interpreter to be used for both source-level debugging and prototyping. The dynamic inputs to this module (appearing after the DUELING step) fill the parameters of the chosen experiment and are described in the next section.

**User interface:** The notation used in the user interface of the PROLOG interpreter to specify the semantics of the state transition is far more
natural than the mathematical notation of the model given in Chapter 6. It is no less precise or powerful, however. While much of the user interface is implied by the discussion in section 8.3.3, a brief presentation of the dynamic inputs will be given in section 8.5.

**Workload Description:** To specify dynamic characteristics of experimental conditions, it was necessary to describe both the properties of the workload and the active events themselves. Since DECSUP works in an event-driven environment, the receipt and transmission of messages characterizes its workload. It is described in this module by probability distributions for both inter-arrival (and inter-departure) periods and size (in number of messages) received or transmitted. Inter-arrival periods are in terms of numbers of phase changes. Workload at each cell is assumed to be periodic, and each cell may be specified to start on any given phase, and have any period.

DECSUP was designed to be a tool for the highest command level. By following the structure of the formal model described in Chapter 6, however, a much higher degree of generality was achieved. Current work includes expansion to other levels of command, and with these results may come the embellishment of either the model itself, or at the very least, of the standard assumptions that a search procedure for consequences can proceed from (see also section 6.3.2).

8.2.4. Verification and validation of rule bases

The PROLOG programming language was chosen as a tool that exemplifies logic programming and logic modeling, and the use of logic modelling is one of the basic ideas that this study seeks to test. As the size of PROLOG programs (or rules bases) grows, problems are being encountered. Therefore a PROLOG program called DUELING was written, which first examines PROLOG rule bases for inconsistencies that might cause problems later on.

Among high-level languages, however, none uses the IF “…” THEN rule format like PROLOG. PROLOG programs may rightly be viewed as collections of rules. In addition, PROLOG implementations that follow the
Chapter 8

so-called Edinburgh syntax (see section 3.2) have a powerful set of predicates that simplify manipulation of the various parts of rules. The more the part of this study relating to verification of development of decision-making algorithms by the use of logic programming was considered, the more it became clear that the types of errors that might be encountered in non-PROLOG rule bases could be easily reproduced in PROLOG, and that a program could be developed to find those errors.

For these reasons, we decided to devise the rule-based checking tool DUELING in PROLOG. Most of the development work on DUELING involved use of AAIS-PROLOG on a Macintosh. LPA-PROLOG was also used on an IBM AT in the interests of portability. A listing of DUELING is found in the appendix.

8.3. The user interface

To present a decision situation in DECSUP, a description format based on broad categories of analysis (Hopson [84]) that are common to most decisions, and that frequently underlie decision-making difficulties, is defined below. The decision situation description format used in DECSUP's user interface is organized in terms of eight categories:

1. Situational objectives

This first part of the description simply documents the results of the goal decomposition. It specifies the high-level goal that provides the common focus to decision-making and cognitive processing in this decision situation.

2. Task dynamics

The second part of the decision situation description defines the kind of dynamic context in which the situation is embedded. There are three choices:
Closed-loop iterative:
A repetitive situation that once completed, must immediately be faced again;

Unfolding:
One in an evolving series of decision situations in which the form of the nth in the series depends strongly on the results of the n-1 th;

Single instance:
A once-and-only-once type of decision that will never be made often enough by a given decision-maker to allow any degree of expertise to develop.

3. Selection criteria

Most decisions are viewed in different ways by different commanders because they use different criteria for evaluating particular choices. One of the difficulties that a commander faces is combining these various choice criteria over a set of alternatives (or even for a single alternative). In many cases this is primarily a knowledge problem, where the best or even good strategies or rules for trading off criteria are unclear and/or unknown.

Identifying the choice criteria that the decision maker must combine and determining how this combination task is understood is important for aiding this part of the decision process.

4. Underlying process

Decision problems usually revolve around some complex real-world process that involves controllable aspects, e.g. the decision-maker's own actions, random or unconcerned elements (e.g. weather, physical laws), and sometimes even hostile elements (e.g. enemies). The evaluation of a potential course of action thus depends on projecting the outcome of that course of action given the underlying process involved. However, commanders have problems in evaluating a potential course of action. First, they often do not understand the underlying process well enough
to be able to predict its behaviour effectively. And even when they do have a deep understanding of the mechanics of the underlying process, they have difficulties in extrapolating events in that process in time and space.

Describing this aspect of the decision situation description must proceed from two perspectives:

- a description of the underlying process that the decision-maker must deal with, and

- the decision-maker's understanding and perception of this process and its dynamics.

5. Information environment

A major difference between today's commander's work and that of a generation ago is the amount of information available. Previously, many decisions had to be made in the face of too little information. Today, most are made in the presence of too much. A great variety of information is available to the decision-maker, but all too often the relevance of each item of data is unclear, and there is insufficient time to give detailed consideration to each one. This can result in the paradoxical case of the really important piece of information being lost or overlooked in the avalanche of irrelevant detail. This part of the situation description details the three kinds of information that the decision-maker must deal with:

- The input elements of information that are perceived as relevant to the decision and that also change dynamically during each instance or iteration through the command and control cycle;

- The parameter elements of information that are perceived as relevant to the decision but that do not change during each instance or iteration through the C2 cycle;

- The output elements of information which the decision-maker creates from the inputs and parameters to help in the decision process or that
the decision-maker must provide to other cells and evaluate as part of the decision.

6. Intermediate analysis

While everyone uses a variety of 'common sense' reasoning skills in making decisions, experienced commanders bring special knowledge, i.e. expertise, to the task. They discover special properties of the decision situation that can help them choose a course of action, so that by reasoning about the decision situation they can clearly determine an optimal decision. These special properties that help guide the unaided decision process are incorporated into the commanders' 'mental model' of the decision situation, so that they form a tight link between the way a commander represents the problem and the structure of his decision process. The effectiveness of these special properties can be improved by computer support, but only if the DDSS designer knows what the commanders' reasoning processes are. Discovering them is the purpose of this portion of the decision situation description.

7. Decision representation

A commander will characteristically have a set of ways of thinking about or representing a decision problem, so that he can better solve it. Chess masters represent a chessboard as having 'zones' which are controlled by white, black, or are under contention. This representation helps them choose moves and relate a specific goal event (capturing the opponent's king) to the current characteristics of the situation. This tendency of thinking about strategic situations in more abstract terms is common to virtually all commanders. It is therefore essential that a DDSS reflect the commander's representation of the problem.

8. Required judgments

No matter how much analysis and probing is done, there may remain a residue of mental tasks that defy precise description. The commander performs these mental tasks, but can't really say how he does them. We may call these ill-defined portions of the decision process 'judgments'.

130
They are explored in the last section of the situation description. The main purpose of this section is to identify those judgments that are required of the commander, and to collect some information on them.

This description format may be summarized into a one-page format that Hopson et al. [84] called the Summary Tabulation of Aiding Requirements, or STAR Table. The STAR Table is a terse, structured summary of the key elements of the decision situation that are relevant to developing a decision-aiding concept for it. A modified STAR Table format is shown in Table 8-1, and an example of use in a specific decision situation is given in Table 8-3.
DECISION SITUATION: (Name or title of decision situation)

TASK DYNAMICS: (Type: closed-loop iterative, unfolding, or single-instance)

SITUATIONAL OBJECTIVE: (Highest-level goal that defines the situation, ideally expressed in terms of the 'goal event' - physical or observable event(s) that the decision-maker is trying to achieve or accomplish)

UNDERLYING PROCESS: (Provide a brief one-sentence description of the observable process in which this decision situation is embedded)

VALUE CRITERIA: (Listing of the individual criteria by which a candidate decision is compared against the goal event; this list specifies the attributes by which the quality of a decision is measured by the decision-maker)

INFORMATION ENVIRONMENT:  
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>(List of individual information items that are available for the decision process and that may change in value during a given instance of the decision situation)</td>
<td>(List of individual information items that are created during the decision process; some or all of these may be aspects of the decision being made)</td>
<td>(List of individual information items that are available for the decision process and that do not change in value during a given instance of the decision situation, but may change from instance to instance)</td>
</tr>
</tbody>
</table>

INTERMEDIATE REASONING/ANALYSIS STEPS:  
(List the intermediate reasoning steps or kinds of analysis that the decision-maker typically applies in the unaided or baseline decision process)

REPRESENTATION:  
(A simple one- or two-sentence description of how the unaided decision-maker represents the decision situation; it should indicate whether the decision-maker's mental model of the situation is framed in words or in pictures, as well as how tight or loose the representation is to the unaided decision-maker)

REQUIRED JUDGMENTS:  
(A listing of the heuristic judgments that the decision-maker must make during some or all instances of this decision situation; the list should focus on those judgments that involve processing of quantitative information and/or uncertainties)

Table 8-1: STAR Table Format
Chapter 8

The decision-presentation process can better be understood by means of a more concrete example. One specific decision situation identified in Table 8-2 is the 'on-station search' situation in an Air Anti Submarine Warfare (ASW) mission.

<table>
<thead>
<tr>
<th>DECISION SITUATION</th>
<th>GOAL EVENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON-STATION SEARCH</td>
<td>GAIN CONTACT WITH TARGET OF INTEREST</td>
</tr>
<tr>
<td>CONTACT CLASSIFICATION/VERIFICATION</td>
<td>IDENTIFY SOURCE OF CONTACT</td>
</tr>
<tr>
<td>LOCALIZATION</td>
<td>DETERMINE LOCATION, COURSE, SPEED, AND DEPTH OF TARGET</td>
</tr>
<tr>
<td>SURVEILLANCE TRACKING</td>
<td>MAINTAIN LOCALIZED CONTACT WITH TARGET</td>
</tr>
<tr>
<td>ATTACK PLANNING</td>
<td>MOUNT OPTIMAL ATTACK AGAINST HOSTILE TARGET</td>
</tr>
<tr>
<td>LOST-CONTACT REACQUISITION</td>
<td>REGAIN AND LOCALIZE CONTACT WITH A LOST TARGET</td>
</tr>
</tbody>
</table>

Table 8-2. ASW decision situations and goal events

During this situation (and, in fact, throughout the entire Air ASW mission), the aircrew attempts to optimize utilization of the sonobuoys (acoustical sensors) available to them. The initial use of sonobuoys is to gain contact with the threat target. After initial contact is gained, the contact is refined through deployment of additional sonobuoy patterns until a high-resolution contact is finally achieved. Although the crew
receives information on suggested initial sonobuoy patterns at its preflight briefing, these patterns are based upon predictions of environmental conditions which may or may not be valid for the search area at the time of the mission. Because of the unreliability of predicted conditions, predetermined sonobuoy patterns may be significantly suboptimal when variations exist between the predicted and actual conditions. The objective of the on-station search decision situation is to determine which combination of pattern geometry, spacing, and placement is optimal to gain initial contact given the actual or in-situ environmental conditions and target uncertainty. The general decision-situation presentation described above (STAR Table) was applied to the on-station search decision situation, resulting in the Table 8-3.

Another example (Figure 8-2) presents the layout of an output screen using the format described above as it is presented by DECSUP to the commander at the beginning of a decision-making process. The graph in the upper right-hand corner presents the Key Tasks relationships. These relationships were analyzed in Chapter 5 and are shown at the same place on each preceding or following screen during the decision-making process. The commander may select a specific Key Task, and the appropriate information is presented. The screen layout is presented for a Decision Key Task, as this layout is the operative one for any further request to DECSUP.

Each Key Task in the graph has its own screen layout. The commander is provided with an overview of the information available, e.g. assessment reports, situation displays, statistics and so on. The commander may choose the level of detail he is interested in, and DECSUP retrieves detailed information from other information systems connected to DECSUP as shown in Figure 8-2.
DECISION SITUATION: ASW On-station search

TASK DYNAMICS: Iterative

SITUATIONAL OBJECTIVE: Selection of optimal sonobuoy pattern given in-situ environmental conditions and target uncertainty

UNDERLYING PROCESS: Zero or more submarines moving in or through search area.

VALUE CRITERIA: Probability of detection of submarine
Expected time until first contact with submarine
Expected time contact with submarine will be held
Time required to deploy pattern
Number of sonobuoys in pattern

INFORMATION ENVIRONMENT:

Inputs
Oceanographic conditions
- propagation loss
- ambient noise
Sensors remaining
Target contact history
Acoustic processor mode

Outputs
Probability of detection

Parameter
Sensors
- type
- capabilities
Available pattern geometries,
Aircraft capabilities
- acoustical emissions
- movement capability
Area of search
- operating area
- restricted area(s)

INTERMEDIATE REASONING/ANALYSIS STEPS:
1. Calculation of in-situ profiles
2. Exclusion of patterns failing to fit mission restrictions
3. Optimization of sonobuoy patterns

REPRESENTATION:
1. Pattern geometry and types and settings of sensors used
2. Steering commands, queueing sequences, and fly-to-points for pattern deployment
3. Target probability area and track data

REQUIRED JUDGMENTS:
Final choice of pattern.

Table 8-3: Example of decision situation representation.


**DECISION**

**DECISION SITUATION**

Decide need for declaration of alert status

**TASK DYNAMICS**

On demand

**SITUATIONAL OBJECTIVES**

Key Mission Component

**INPUTS: MAINTAIN STATUS ASSESSMENT PLANNING**

Reports, Maps, Recommendations

**OUTPUTS:** Orders, Measurements, Directives, ...

Figure 8-2: Sample output screen

**8.4. Purpose of DECSUP**

The main purpose of DECSUP is to support a war headquarters commander in making a decision. DECSUP's general mission is to recommend courses of action to decide on, and includes, for example, the functional areas of crisis management, reinforcements, and logistics.

DECSUP was placed in a (NATO-CCIS)⁹ environment (Figure 8-4) that

---

⁹ NATO Command and Control Information System: An integrated system comprised of doctrine, procedures, organisational structure, personnel, equipment, facilities and communications, that provides authorities at all levels with timely and adequate data to plan, decide, direct and control their activities.

---

136
bases decision support capabilities on distributed database applications. The main purpose of the NATO Command and Control Information System (NATO CCIS) is to enhance the wartime command and control process by supporting the performance of a staff action that is essential to permit mission accomplishment during tension or war, or an action that directly contributes to the war-making capabilities of a command.

The information systems requirements for the NATO-CCIS include flexibility, survivability, security, interoperability, reliability and transportability. The implementation of these characteristics leans toward small, modular units capable of supporting a wide variety of NATO-CCIS configurations. The modular units must be easy to add or remove from the System. Such features and capabilities are potentially inherent in distributed information system architectures.

In order to meet the requirements specified in the NATO-CCIS Master Plan, distributed information system architectures show multiple levels of physical distribution (Figure 8-3), from geographical distribution over a theater of operations to internal distribution within a single computer system. The system architectures are characterized by:

- Distribution of processing resources (i.e., hardware/software (HW/SW) and data) to locations within an applications system where they can significantly improve the system's performance, reliability, availability and survivability.

- Distributed HW/SW control and management to increase availability/reliability and survivability.

- Distributed database management of functionally partitioned and replicated data to increase performance, reliability and survivability.

- Communications architectures and protocols to support message structures for inter-processor communications, database management, system control and interaction with other systems and networks.
Figure 8-3: Different levels of architecture distribution

Legend:

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASOC</td>
<td>Air Sector Operation Center</td>
</tr>
<tr>
<td>TAC HQ</td>
<td>Tactical Air Command Headquarters</td>
</tr>
<tr>
<td>LAN</td>
<td>Local Area Network</td>
</tr>
<tr>
<td>CRC</td>
<td>Central Region Command</td>
</tr>
<tr>
<td>ArmyTOC</td>
<td>Army Tactical Operation Center</td>
</tr>
<tr>
<td>CRP</td>
<td>Central Region Post</td>
</tr>
<tr>
<td>TACC</td>
<td>Tactical Air Command Cell</td>
</tr>
<tr>
<td>TACP</td>
<td>Tactical Air Command Post</td>
</tr>
</tbody>
</table>
Chapter 8

- Adaptability to changing system size, workload and technology requirements through ease of addition, replacement or exchange of modules in the system.

The NATO Command and Control Information System (NATO CCIS) development was a project combining rapid prototyping based on minimum, or thin, specifications with frequent operational deliveries of subsystems to commanders. Feedback from user, prototype demonstrations, and operational exercises was a primary means of augmenting and validating system requirements for functionality. For individual software elements, the level of formality and detail applied during the requirements phase and the specific steps executed during the implementation phase are tailored to the overall system based on their importance and risk.

The distributed NATO-CCIS (Figure 8-4) is currently operational in Europe at more than 35 geographically dispersed sites. Each site, or node, contains a local-area network interconnecting NATO command and control systems or personal computer work stations. A packet-switched wide-area network interconnects the nodes. User nodes contain analyst consoles, local or remote data-entry terminals and local-area network servers. System nodes provide network-wide resources such as database servers and system management functions.
Figure 8-4: The NATO-CCIS top-level data flow bases decision support capabilities on distributed database applications.

Legend:

NATO-CCIS             NATO Command Control and Information System
NUC/CHEM               Nuclear/chemical operations
DB                     Database
TBD                    To be determined
WWMCCS                 World-Wide Military Command and Control System
Since early definition of a secure network architecture for a CCIS is a key part of any development approach, any system design concept has to optimize security in spite of there currently being no entirely satisfactory solution.

Multi-level secure (MLS), multi-functional systems are a long way from achieving recognition, hence the full benefits of the architecture would be constrained by the need to operate individual processors at a ‘system high’ classification with trusted computer sub-systems. This would not unduly detract from the philosophy for maximum decision support distributed to staff cells if the local communication system could ensure at least a separation of classified and unclassified information in the volatile tactical environment. However, special precautions need to be taken in peacetime to restrict access to sensitive information.

8.5. Data integrity and security

During recent months the world has experienced rapid and radical changes in the politico-military situation that have led to a new security environment. In a changing environment like the present one with its ill-defined threat, less clear warning indicators, and more difficult decisions on necessary actions, the need for security will grow in importance. New requirements resulting from, e.g., verification of arms control agreements, could pose new challenges to the security of C2 systems.

In order for DECSUP to function securely in a military environment, the integrity of the data transmitted over the communication links between cells must be maintained. The use of either link or end-to-end encryption will help to maintain the integrity of transmitted data, although there are differences in the level of security that they provide. This section begins with an analysis of link and end-to-end encryption and ends with conclusions on communication security having direct impact on the implementation of DECSUP.
8.5.1. Introduction

Data-communication security is typically analyzed using the International Standards Organization (ISO) Model of Architecture for Open Systems Interconnection (OSI). In this model, the data-communication path is logically composed of an ordered set of subsystems, called layers, through which application programs communicate (Voydock and Kent [85]). Each of the seven layers (physical, data link, network, transport, session, presentation and application) is composed of protocols. Peer-to-peer protocols provide communication between layers at the same level (i.e., between two transport layers). Interface protocols are the layer-to-layer connection rules that describe how each layer relates to its neighbouring layers (i.e. how the presentation layer relates to the session and application layers).

In a distributed network, data must be transmitted over communication links whose physical protection may be either uneconomical or infeasible. For such a network to function securely, the integrity of the transmitted data must be maintained.

Data encryption is the fundamental technique upon which all communication security is based. There are two basic ways in which data encryption may be applied to networks: link encryption and end-to-end encryption. These two approaches differ in the nature of the security that they provide.

8.5.2. Link encryption

Link encryption provides security for message transmission over an individual communication link between two network nodes, regardless of the sources or destination of the message. The message may be transmitted over many communication links before arriving at its final destination, and in many cases, the links will be physically unprotected and subject to attack [85]. Within each node, incoming messages are decrypted, and then encrypted again using the key for the next link that is to be used to forward the message (Turn [86]).
Since the transmitted message is decrypted and encrypted at each node through which it passes, all data flowing on the links, including the final destination addresses, are encrypted (Diffie and Hellman [87]). Link encryption therefore masks the origin-destination patterns. In addition, if there is a continuous flow of information between the nodes, it masks message frequency and length characteristics (Rutledge and Hofman [88]). Link encryption requires that all nodes be physically secure, since the compromise of any network node will expose substantial amounts of information. However, it is expensive to maintain security at each individual node. In addition to the one-time expense of providing encryption hardware and a secure physical environment for each node, there are a number of ongoing expenses in NATO whose total cost may exceed the one-time outlays. These include the cost of the employees to protect the physical security of the nodes, the cost of frequent audits to ensure that security policies and procedures are being carried out correctly, and the cost of insuring against losses resulting from security violations [85]. Since it is difficult to fairly apportion these costs among the network users, the overall cost is typically amortized over all the users, whether or not they feel the need for such protection.

In conventional encryption algorithms, the key used to decipher a message is the same as that used to encipher it. Such a key must be kept secret and should be changed on a regular basis. Key distribution is a problem in link encryption systems (Gait [89]). Since each node must store the encryption key of every link to which it is connected, either physical transportation of the keys must be used, or special network facilities must be established [86]. The wide geographical dispersion of the network nodes complicates the process and increases the cost of continuous key distribution.

8.5.3. End-to-end encryption

While link encryption techniques model a network as a collection of nodes joined by communication links, each of which may be independently protected, end-to-end encryption techniques model a network as a medium for transporting data in a secure fashion from source to destination. Since a message is protected throughout its
transmission, it will not be compromised if a node has been subverted. As a result, end-to-end encryption provides a higher level of security in a network environment because the message is not deciphered until it reaches its final destination (Denning [90]).

End-to-end encryption systems are significantly less expensive, more reliable, and much simpler to conceive, implement and maintain than link encryption systems [89]. Furthermore, end-to-end encryption is more naturally suited to the military perceptions of security requirements since only the source and destination nodes must be secure. Individual users may choose this method of encryption without affecting other users in the network, and thus the costs of end-to-end security may be more accurately apportioned [85].

End-to-end encryption does not permit message addresses to be encrypted, since each node through which the message passes must have access to the address to decide how to forward the message. This method of encryption is therefore more susceptible to attacks of traffic-flow analysis since the origin-destination patterns are not masked.

8.5.4. Conclusion and recommendation

Communication security in distributed systems may be threatened by passive attacks, in which intruder listens to the communications passing on a connection, and active attacks, in which an intruder interferes with the communications passing through a line. Passive attacks, such as the release of message contents and traffic analysis, can be effectively prevented, although they typically cannot be detected. Active attacks, such as message stream modification, denial of message service and spurious association initiation, can be detected, but not prevented.

The use of encryption will prevent passive attacks by masking the data patterns and transforming the plaintext data into a form which is generally unintelligible. Active attacks can be detected through the use of an encryption algorithm with appropriate error-propagation characteristics, as well as through the frequent exchange of message blocks between peer connections. The integrity of transmitted data may
be maintained through the use of either link encryption or end-to-end encryption. However, end-to-end encryption is believed to provide a higher level of security in a network environment.

In the short term security can be achieved in a communication system by the expedient of two LANs, one for classified and one for unclassified information. However, most staff cells would have a need for terminals on both LANs to operate concurrently and hence TEMPEST could not be significantly improved over today’s arrangement (Figure 8-5).

Figure 8-5: Communications security
In the longer term, security on LANs will far exceed the levels of security currently provided on local information distribution layouts. Data security techniques are improving rapidly, driven by commercial needs, especially banking. TEMPEST security will be significantly improved by the predominance of fibre-optic cabling and the inherent characteristics of a digital LAN. For example, at any one point on the LAN, information concerning only one application is present, and the LAN therefore provides a time division system with inherent electromagnetic separation. However, this simplistic picture belies many problems which have to be overcome for even a dual-level secure LAN to achieve accreditation.

The separation of classified from unclassified information, and indeed of multi-level security classification separation, at the gateway to the external communication systems can be achieved by the use of 'front-end' mini-computers, certain of which have already been evaluated for this specific task.

The next Chapter provides a general discussion on evaluation and presents some findings gained from the use of DECSUP.
9. EVALUATION

In this Chapter we discuss the evaluation of DECSUP. A number of alternative methodological approaches will be profiled and juxtaposed in this Chapter. The barriers to DECSUP's successful implementation are reported and the evaluation results are presented as attempted thus far.

DDSS are characterized by the complexity of their architecture, involving interactions between heterogeneous, distributed components (cells, headquarters, operational units, etc.). The evaluation of such systems is not easy, and information on efficiency is often obtained only after implementation and tests in the target environment (Sainfort et. al. [91]). This aspect is particularly critical for criteria such as vulnerability or timeliness and requirements related to performance.

But much more important than a specific evaluation result is the need to recognize that, as Riedel and Pitz [92] emphasize, DDSS evaluation provides information to the design team, users, and decision-makers involved. Information is useful only if it shapes decisions; therefore, evaluations should be made in light of the time line associated with decisions to be made with respect to the system under development. Timely evaluation information is therefore crucial.

9.1. Introduction

We encountered a number of problems in trying to identify Decision Key Tasks for which we could usefully evaluate DECSUP with respect to effectiveness and efficiency. It is likely that many of these difficulties will be present in other complex decision support systems. Measuring the effectiveness of particular decisions were difficult because there were always many background variables that affect the result. In particular Decision Key Tasks it was not very clear what constitutes success. Even in a Key Task such as naval air defense where success is clear-cut (i.e. the ship avoids being hit by a missile), it was clear from observing the appropriate exercise that some strategies are riskier than others, and it takes an expert to detect the difference. Disagreements can arise even
between commanders as to what constitutes a good strategy.

The main sources of difficulty in identifying appropriate Decision Key Tasks were as follows:

- Conflicting accounts of the decision-maker's task. (There were conflicting accounts of what the Decision Key Tasks entailed. Different commanders had different ways of performing the job and different aspects that they felt to be important for decision-making.)

- Difficulty in changing the way of working. (This is because if particular individuals are inexperienced at their jobs then the commander has to provide support for them and has not enough time to work with the new system)

It is unlikely that these type of problems are unique to this particular system. The problems of the commander constitute what Ackoff [93] calls a 'mess'. These difficulties were the reason for the decision to run the evaluation of the system twice. The intention is to resolve the conflicts between the different interpretations of certain Decision Key Tasks, and increase the level of experience in the execution of these Key Tasks.

9.2. Alternative approaches to evaluation

In the initial stages of the study, a literature review was conducted to examine existing approaches for evaluation of decision support systems. The emphasis of much of the DSS literature was on three primary methodologies that have been used repeatedly for evaluating decision support systems: Cost-benefit analysis (CBA); value of information approaches (drawing on concepts and assumptions from information economics); and multiattribute utility (MAU) models. We shall review each in turn.
9.2.1. Cost-benefit analysis

Cost-benefit analysis (CBA) is a seemingly logical choice: group, measure, and then compare costs and benefits accruing from a proposed system or off-the-shelf package and make a decision on the basis of the readings on the balance sheet. Extensive work has been done on identifying the potential range of benefits and costs accruing from a decision support system.

Possible DSS benefits include items pertaining to cost reduction or avoidance, error reduction, increased flexibility, increased speed of decision-making, and improvement in planning or control. A generic list would include the following:

- Benefits from contributions to record-keeping tasks;
- Benefits from contributions to record-searching task;
- Benefits from contributions from DSS restructuring capability;
- Benefits from contributions of analysis and simulation capability;
- Benefits from contributions to decision-making process and resource control.

Analogously, possible DSS costs can also be catalogued in the abstract:

- Procurement costs;
- Start-up costs;
- Project-related costs;
- Ongoing costs.

The characteristics of DSS that affect cost and benefits must be identified. One typology takes into account:

- Accuracy;
- Response time;
- Security;
- Reliability;
- Flexibility.
A more complete scheme is presented in Martin and Trumbly [94] that specifies discrete performance indicators for the criteria of accuracy, response rate, efficiency, and user accommodation across the factors of data entry, report processing, system change process, system environment and decision effectiveness\(^\text{10}\). Note that decision effectiveness is critical.

How has CBA fared? It has been used in countless system acquisition and ongoing evaluation exercises. The results have been disappointing: one expert cautions against using it at all Lay [95]. To cite just one horror story, for one project, eight models were developed to estimate development costs; the answers ranged from $362,000 to $2,766,000 (Fripp [96]).

The latter result should underscore the point that costs are difficult to determine. Other problems with, and obstacles to, CBA include the difficulty of evaluating system effectiveness, the fact that intangibles are important but cannot be monetized, and the reality that measuring the cost of user involvement is hard to accomplish. In a more ultimate sense, costs and benefits may be incorrectly identified or incompletely specified.

More generally, it can be concluded from the discussions above that the cost-benefit approach is simply not applicable to evaluations of decision support systems. Such systems must meet a host of software-engineering-related standards, but a DSS is expected to satisfy a set of more rigorous and conceptually complex assessment features related to the decision process and outcomes.

In one very fundamental respect, the cost-benefit approach leads very naturally to a value of information. As Keim and Janaro [97] point out, the key benefit of any information system (including, of course, DSSs) is the value of the information that it makes available to the user. Any item of information is needed if a decision would be different without it, warranting the inference that such an item of information has no value if it cannot influence a decision. Decisions are made all of the time with imperfect information because the necessary information is unavailable.

\(^{10}\) Decision effectiveness: The contribution of the DSS's output to the enhancement of the efficiency and/or quality/effectiveness of the user's conclusion or the commander's or group's decision.
Chapter 9

The effort to acquire the information may be prohibitively costly, there is no knowledge of the information's availability, or the information is not available in the form required. Consequently, additional or different information always has value (and should be provided - via the DSS - if the costs are low enough).

9.2.2. Value of information

The second methodology is the assessment of the value of information as an attribute to the output of a decision support system. This approach reflects the reasonable premise that the key benefit of a DSS is the value of the information that it makes available. As noted above, decisions made on the basis of imperfect or flawed information inputs occur because:

- the required information is unavailable (such as insight about an enemy's strategic intentions);

- the requisite effort to obtain it would be too costly (for example, definitive information about an opponent's troop deployments in a very low-priority area);

- there is no knowledge of the availability of the desired information (such as the enemy has a communications channel unknown to the interested observer);

- or the information is not available in the form needed (a good illustration is the knowledge of a force's emanations without the ability to determine their content).

Information is of value to the extent that it makes the base of knowledge less imperfect - if the cost of obtaining the information is outweighed by its value. Approaches to the 'value of information' concept range from theoretical utility-based orientations to Bayesian (expected value) techniques.
For determining the value of information, it is necessary to specify measures for several components of a theoretical framework (Feltham [98]). For all decisions involved, there is a need to specify all known alternatives. To evaluate the impact of DSS changes, the decision-maker must have a payoff function over events that occur as time unfolds.

DSS evaluation in this sense can occur only vis-à-vis alternative DSS's - including the case where the alternative system is either the status quo or the case where the system supplies no information at all. While the demanding theoretical inputs of the 'value of information' approach cannot be satisfied exactly and comprehensively in the DSS domain (e.g. a military headquarters), this emphasis on a comparative perspective does, however, ingrain a very useful and quite practicable guideline into the discussion of DSS evaluation.

The 'value of information' approach treats the potential contribution of a DSS in terms of several distinct criteria. One is relevance, the idea that messages are produced only if the cost of doing so is less than the value. A message from the DSS may be irrelevant, it should be noted, if events that it describes can be accurately estimated given other messages received.

Timeliness refers to the fact that the timing of the receipt of information may affect the payoff to the decision maker. Reporting delay is the interval between the time of the event and the time that the message is received. Ideally, the reporting interval will match the decision interval. Decreases in the reporting interval usually require changes in the DSS, which increase costs. The value of the decreased delay is expected to exceed the cost involved.

Accuracy reflects the fact that errors in the measurement or processing of data generate differences between messages and events. Errors in the DSS cause uncertainty to be felt about past events. Uncertainty regarding past events causes more uncertainty about future events. This in turn can be expected to lead to lower-quality decisions.
Chapter 9

For determining the value of information, it is necessary to specify measures for several components of a theoretical framework (Feltham [98]). For all decisions involved, there is a need to specify all known alternatives. To evaluate the impact of DSS changes, the decision-maker must have a payoff function over events that occur as time unfolds.

DSS evaluation in this sense can occur only vis-à-vis alternative DSS’s - including the case where the alternative system is either the status quo or the case where the system supplies no information at all. While the demanding theoretical inputs of the ‘value of information’ approach cannot be satisfied exactly and comprehensively in the DSS domain (eg. a military headquarters), this emphasis on a comparative perspective does, however, ingrain a very useful and quite practicable guideline into the discussion of DSS evaluation.

The ‘value of information’ approach treats the potential contribution of a DSS in terms of several distinct criteria. One is relevance, the idea that messages are produced only if the cost of doing so is less than the value. A message from the DSS may be irrelevant, it should be noted, if events that it describes can be accurately estimated given other messages received.

Timeliness refers to the fact that the timing of the receipt of information may affect the payoff to the decision maker. Reporting delay is the interval between the time of the event and the time that the message is received. Ideally, the reporting interval will match the decision interval. Decreases in the reporting interval usually require changes in the DSS, which increase costs. The value of the decreased delay is expected to exceed the cost involved.

Accuracy reflects the fact that errors in the measurement or processing of data generate differences between messages and events. Errors in the DSS cause uncertainty to be felt about past events. Uncertainty regarding past events causes more uncertainty about future events. This in turn can be expected to lead to lower-quality decisions.
Chapter 9

King and Epstein [99] present a more exhaustive specification of the relevant criteria of information value.

Rouse [75] considers the value of information from a more conceptual perspective, asking what it is about the term that makes it of interest to DSS designers. First, he notes, the 'value of information' is an elusive concept. Ordinarily, value is thought of as what one is willing to pay in money and/or effort. This definition, however, is not of direct utility for defining information requirements. The concept must be tapped in terms of *why* a user would be willing to pay for information.

One reason is to reduce uncertainty. Uncertainty can be reduced by decreasing the variance of a user's estimate of one or more variables. The help functions of a decision support system may serve this purpose, as could the use of formal modelling routines and sensitivity tests (via the DDSS model base) and/or the retrieval and manipulation of data stored in the system's database.

If information provided is relevant to the task at hand, it increases in value. Further, information's value is closely tied to its availability at the point in time that it can exert an impact on decision-making. Appropriateness of form is a third aspect. Information can be such that it reduces uncertainty and is relevant, but lacks value because of its form. Rouse cites the example of a repair manual written in a foreign language. Analogously, a DSS/user interaction in a *jargon* - unfamiliar to the user will not be used.

An alternative in the 'value of information' approach is an empirical strategy, which is based on observational data. This approach is based on 'real data', and has yielded an empirically grounded topology of information attributes that has gained general acceptance in the literature (Sol [36]). It does have the virtue of generating precise and direct inputs from potential users of a DSS about their perceptions of the information structure of the domain.
9.2.3. Multiattribute Utility (MAU) Assessment

We provide in this Chapter an example of a MAU-oriented DDSS evaluation and report the results of assessing DECSUP. MAU assessment is ideal for this example because of the reliance on an analytical framework comprised of the nodes DDSS, user, decision-making organization, and environment.

The basic idea of MAU is that evaluation must take into consideration relevant linkages involving the nodes stated above. For example, the performance of the DDSS/user-interaction must be acceptable. But this alone is insufficient. A second interaction - linking the user and the larger decision-making organization - also should be considered. Does the DDSS facilitate the organization's decision-making process? The third interaction is the one between the decision-making organization and the environment.

Measures

Measures of effectiveness can in turn be identified in relation to the three interactions identified above. Each measure should be (objectively or subjectively) measurable and must correlate (positively or negatively) with the DDSS's overall effectiveness.

The measures of effectiveness for each interaction are listed in the tables 9-1, 9-2, and 9-3 respectively and are organized hierarchically. The three uppermost levels represent the three interactions. Each of the three top-level interfaces is subdivided for the purpose of isolating distinct and measurable measures of effectiveness. The task of evaluating a DDSS is thus transformed into one of assigning scores to the bottom-level nodes of the evaluation hierarchy.

Measures assessing the effectiveness of the DDSS/User Interaction (Table 9-1) are divided into two distinct clusters: one that assesses the match between the DDSS and the user's background and operational needs, and those that assess the adequacy of the DDSS's characteristics (ease of use, response time, adequacy of the data files, and so forth).
Measures which index the effectiveness of the User-DDSS/Decision-making organization interaction (Table 9-2) are also divided into two major groups of criteria: those assessing the DDSS's efficiency from an organizational perspective, and those that assess the system's fit with the organization. The former includes such characteristics as the amount of time required to accomplish the Decision Key Task with the DDSS (a DDSS, one might think, would save time, but this is in fact not the only empirical outcome from the use of a support system - it often lengthens the time required), data management and set-up time requirements, and the system's perceived reliability and supportability under typical conditions of use (in this case, under exercise conditions). To assess the DDSS's fit with the organizational context, there are criteria focused on the potential effect of the DDSS on organizational procedures, other people's work, the flow of information, and its value in performing other Key Tasks.

Finally, measures of effectiveness (Table 9-3) assessing the quality of the decision-making organization/environment interaction are grouped into three categories of criteria: the perceived quality of decisions reached via the use of the DDSS, the extent to which the DDSS's technical approach matches the technical requirements of the task and the extent to which the DDSS improves the quality of the decision-making process.
<table>
<thead>
<tr>
<th>DDSS/User interface</th>
<th>Match between DDSS and personnel</th>
<th>Match with training and technical background</th>
<th>Match with workstyle, workload, and interest</th>
<th>Match with operational needs</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.81</td>
<td>7.93</td>
<td>8.44</td>
<td>7.08</td>
<td>8.27</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DDSS's characteristics</td>
<td>General characteristics</td>
<td>Ease of use</td>
<td>8.56</td>
<td></td>
</tr>
<tr>
<td></td>
<td>7.70</td>
<td>Transparency</td>
<td>7.78</td>
<td></td>
</tr>
<tr>
<td></td>
<td>8.46</td>
<td>Ease of training</td>
<td>8.83</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Response time</td>
<td>8.67</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Specific characteristics</td>
<td>User interface</td>
<td>7.67</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6.95</td>
<td>Expert judgment stored in DDSS</td>
<td>6.84</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ability to modify judgment</td>
<td>6.88</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>DDSS's automatic calculations</td>
<td>7.16</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>DDSS's graphs</td>
<td>5.84</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>DDSS's text</td>
<td>7.33</td>
<td></td>
</tr>
</tbody>
</table>

Table 9-1: Subjective evaluation mean scores on DDSS/User interaction
<table>
<thead>
<tr>
<th>User-DDSS/organization interface</th>
<th>Efficiency factors</th>
<th>Time required for task accomplishment</th>
<th>7.11</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Time required for data management</td>
<td>6.20</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Set-up requirements</td>
<td>6.33</td>
</tr>
<tr>
<td>Match between DDSS and organization</td>
<td>Effect on organization procedures</td>
<td>7.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Effect on other people's position</td>
<td>Political acceptability</td>
<td>7.67</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Other people's workload</td>
<td>7.17</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value in performing other tasks</td>
<td>7.50</td>
</tr>
<tr>
<td></td>
<td>Side effects</td>
<td>Value to other services</td>
<td>8.33</td>
</tr>
<tr>
<td></td>
<td>8.50</td>
<td>Training value</td>
<td>8.67</td>
</tr>
</tbody>
</table>

Table 9-2: Subjective evaluation mean scores on User-DDSS/organization interaction
<table>
<thead>
<tr>
<th>Organization-environment interface</th>
<th>Decision quality</th>
<th>7.58</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.09</td>
<td>Technical soundness (match between DDSS’s techn. approach &amp; analysts’ techn. requirements)</td>
<td>6.52</td>
</tr>
<tr>
<td>Decision process quality</td>
<td>Framework incorporating judgment</td>
<td>6.83</td>
</tr>
<tr>
<td>7.18</td>
<td>Survey range of alternatives</td>
<td>7.33</td>
</tr>
<tr>
<td></td>
<td>Survey range of objectives</td>
<td>6.83</td>
</tr>
<tr>
<td></td>
<td>Weighting consequences</td>
<td>7.33</td>
</tr>
<tr>
<td></td>
<td>Assessment of consequences</td>
<td>6.67</td>
</tr>
<tr>
<td></td>
<td>Reexamination of decision-making process</td>
<td>7.11</td>
</tr>
<tr>
<td></td>
<td>Use of information</td>
<td>7.78</td>
</tr>
<tr>
<td></td>
<td>Implementation</td>
<td>7.50</td>
</tr>
<tr>
<td></td>
<td>Effect on group discussions</td>
<td>7.11</td>
</tr>
<tr>
<td></td>
<td>Confidence</td>
<td>7.33</td>
</tr>
</tbody>
</table>

Table 9.3: Subjective evaluation mean scores on Organization/environment interaction.

Method

The measurements were constructed by evaluating the responds to the questionnaires distributed during the work described in [2].

Respondents concerning DECSUP questionnaires were asked to rate each of the measures shown at the tables 9-1, 9-2, 9-3 on a scale from 1 to 10 of...
significance. The approach to the construction and use of the questionnaire was based on Gael's [100] approach, except that he derives the list of measures from group discussions with experienced members of the target organization. An extension of the approach that was made in this study was that respondents were asked to rate measures within the context of two different scenarios (see also section 9.1).

The questionnaires were administered to 19 serving officers, 16 of whom were currently working, or had worked, as commanders. The other 3 had had very closely related experience. They had done similar jobs and were involved in training courses for commanding officers.

Results

Means of ratings for the 33 items are given in Tables 9-1, 9-2, and 9-3. The 33 items had a total mean effectiveness rating of 7.367, with a standard deviation of 1.523. The correlation between the samples' total scores at the first scenario and at the 3-week retest for the second scenario was a highly significant value of .97 (p < .001), indicating high measurement stability. The internal consistency was determined by the Cronbach alpha test, Cronbach and Meel [101], applied to the overall measures. The resulting Cronbach alpha reliability index of .815 was above the minimum value (.80) recommended by Brown [102] for such a test. Therefore, both reliability tests indicate that this evaluation is stable and should provide consistent results.

The DECSUP/User interaction received the highest mean score, 7.81, of the three interactions. The comparable scores for User-DECSUP/organization and the Organization/environment interaction were 7.11 and 7.09 respectively.

<table>
<thead>
<tr>
<th>Items</th>
<th>Mean</th>
<th>S.D.</th>
</tr>
</thead>
<tbody>
<tr>
<td>DECSUP/USER interaction</td>
<td>7.81</td>
<td>1.42</td>
</tr>
<tr>
<td>User-DECSUP/organization interaction</td>
<td>7.11</td>
<td>1.94</td>
</tr>
<tr>
<td>Organization/environment interaction</td>
<td>7.09</td>
<td>1.64</td>
</tr>
</tbody>
</table>
Calculating the consistency of responses was used as a method of validation of the questionnaire results. Although the variance of responses as a proportion of the mean was quite high for each of the interactions and each of the scenarios, the ordering of the Decision Key Tasks for each of the measures was consistent. Kendall's Coefficient of Concordance (Siegel [103]) produced highly significant agreement for the ranking of occurrences of behaviour within each interaction and scenario, with \( p < 0.001 \) in each case.

9.3. Summary

The numerous results discussed in detail in the previous section reflect the system's quantitative effectiveness. While these numerical values bespeak the high effectiveness of the system under observation, several important findings were made only during the evaluation phase. These are:

Despite the profusion of DDSS's, intelligent terminals, teleconferencing, and information systems, higher commanders continue to rely to a great extent on informal, noncomputerized information when they make decisions. They believe computers do not provide much information directly to a higher-level commander. Under conditions of high uncertainty, higher-level commanders are more likely to prefer verbal as opposed to written media - a finding with important implications for crisis decision-making in military headquarters.

Further discussion with commanders establish the fact that decision support systems play a minor direct role in the information-gathering network of higher-level commanders. However, the scheduled briefing for discussions of consequences of alternative decisions was rated very highly in the richness of information provided and emerged as the favored medium for all of an executive's different decision roles.

More generally, the discussions point to the need to assess the value of information from multiple sources empirically and comparatively. Information provided - from DDSS and other sources - should not be insufficient in quality or quantity. However, the volume of information
should also not be excessive. Finding the appropriate amount is the key.

The discussions among users of DECSUP also suggest that, while overall satisfaction with DECSUP is quite high, there are discernible differences among aspects of the design and actual use of DECSUP for strategic planning. Best fulfilled according to users was the objective of promoting attention to external facets of planning. Also ranked high were assessments of the system's perceived capability and its use of strategies. In contrast, some users were relatively dissatisfied with the system's degree of attention to internal facets in planning, the extent of coverage and integration of various functional areas.\footnote{Because evaluation considered selected Decision Key Tasks only. See discussions on difficulties in selecting Decision Key Tasks at the beginning of this Chapter.}

For now, DECSUP is being used, and has been rated as an excellent training tool for the planning phase of the Command and Control cycle within a military headquarters. The proposed algorithm for calculating consequences of events has shown its strength of promoting attention to external facets of strategic planning. It has proven to improve the quality of recommended alternative decisions significantly.
10. DISCUSSION

The objective of this study was to find ways of reducing the element of uncertainty in decision-making in a distributed environment, as part of an extensive effort to bring all computing assets of a large military headquarters into a unified framework.

Distributed decision-making requires a broad range of information. This information should be primarily external, timely and appropriate. In order to provide this information it was decided to design and develop a prototype of such a distributed decision support system to demonstrate the validity of the theory.

In this Chapter we will present the individual elements of the theory, and discuss the aspects used to develop the prototype with respect to foundations, models, and procedures.

The aspects which will be analysed below are:

- Logic modelling;
- Goal decomposition;
- Distribution;
- Planning;
- Conflicting roles;
- Human operators;
- Reliability.

Together these aspects form the subject area of this study. Each of them leads to an essential element in the development of the prototype presented herein. The experience gained in working with these elements will be summarized in the next section, and recommendations will be made for future research.
10.1. Review

Logic modelling

A fundamental line followed in this study is to show that methodologies based on logic modelling can be developed for functionally accurate, secure and cooperative distributed decision support systems in military environments. The distributed algorithms and control strategies of such systems should function with both inconsistent and incomplete data. These algorithms and strategies are used to extend the range of applications that can be effectively implemented in distributed environments.

There are two ideas that form the basis of logic modelling in relation to distributed decision-making:

1) To view a logic model of a distributed system as a network of cooperating cells that share common goals, where each cell can perform significant local processing using incomplete and inconsistent data,

2) Distributed decision support systems operate in computer network environments where components failures are inevitable during normal operation. Failures not only threaten normal operation of the system, but they may also destroy the consistency of the system by direct damage to the communication links.

A distributed decision support system that is designed in view of these ideas should be able to provide the decision-maker with timely, relevant and meaningful information.

Analyzing distributed decision support systems in relation to this statement requires some perception of which features of their design are most important to their operation. To some extent, that perception may come from existing research or direct experience with particular systems. As a first approximation, these features can be divided into those relevant
to the behaviour of users acting alone, those relevant to the interaction of users with computers, those relevant to the interaction of multiple users and those relevant to organizational behaviour - all within distributed decision support systems. A sampling of such features follows, drawn from discussions with users of DECSUP.

**Goal decomposition**

It was noted in Chapter 3 that human behaviour is goal-directed, and decision-making is no exception. When a decision is made, the user does so to help achieve some specific goal. The initial step in analyzing the decision problem is therefore to discover and map out the decision-maker's goals. This goal decomposition is based here on a mix of standard analysis (of documents and procedures) and interactive analysis (interviewing). In this study decision-making is treated as a pragmatic activity, with Decision Key Tasks defined and interpreted as concrete goals.

In Chapter 5, a methodology has been suggested, which breaks each Decision Key Task down into subtasks in accordance with the C2 cycle. Each breakdown of a Decision Key Task is presented as a directed graph that depicts the information flow between the subtasks. The subtasks are described by 'minispecs' and a data dictionary. Every 'minispec' consists of rules describing how to perform a subtask.

The breakdown of Decision Key Tasks is dynamic during the phase of DDSS development, but static after completion of this phase.

**Distribution**

The individuals in cells in a distributed structure are meant to have a shared concept of their mission (i.e. objectives and situation) at a fairly high level of generality that nonetheless allows them to function effectively in the restricted environment for which they have more detailed knowledge. Achieving this goal is in part a matter of training, so that distributed users share certain common conceptions, and in part a matter of distributing current information, so that they stay in touch
conceptually. Insofar as it is impossible to tell everybody everything, the
system designer needs to know what is the minimal level of explicit
sharing needed to ensure adequate convergence of views. Such
knowledge guides the determination of the capacity needed for
communication links, the fidelity needed for those links, and the
protocols for using them.

In Chapter 6 we have discussed this knowledge in relation to
effectiveness of decision-making. We have defined several strategies such
that the number of messages (quantity of information) is directly related
to the efficiency of the decision-making process.

The main points in defining these strategies are:

1) An analysis of the organization topologies for which each strategy is
appropriate;

2) The use of uncertainty and the related best worst-case decision
strategy to bridge the gaps in information provided from other cells
about future capabilities;

3) The formulation of a distributed dynamic decision problem using
only knowledge of the organization structure local to each cell.

Planning

If systems are known to be imperfect, it is incumbent on their designers
to convey that information. A fairly bounded problem that came up
several times during the study was how to display information about
planning a decision. In the design of existing DSS (Te'eni [104]), it was
implicitly assumed that planning the decision-making process was
manual. In our formulation, DECSUP supports humans in planning this
process by representing the Decision Key Tasks through the STAR
format. That means that a DDSS using this format can help to track the
execution of sub-tasks by providing control feedback. Following the C2
cycle ensures that all sub-tasks are indeed executed and do not get lost
between the decision-maker and the DDSS (see also figure 8-2).
Conflicting roles

All military headquarters face conflicting demands. For example, they may have to act in both crisis and routine situations; they may have to maintain a public face quite at odds with their internal reality (e.g., exuding competence when all is chaos underneath); or they may need to adhere to procedures (or doctrine) while experimenting in order to develop better procedures. Each of this conflicting roles may call for different equipment, different procedures, different personnel, different decisions, different patterns of authority. Mediating these conflicts driven by events whose order is not known before, is essential to organizational survival.

The purpose of section 6.3 was to isolate and then analyze the problem of reasoning about partially ordered events. The outcome is an algorithm that computes useful bounds on necessary and possible consequences of given decisions.

Human operators

No system in a complex, dynamic environment such as a military headquarters works exactly as planned. That is why there are still human operators, even in cases in which actions are executed by machine. (see Chapter 7 for discussion on levels of automation) The role of these operators must therefore come from knowing things that are unknown to the machine. Perhaps because its formalized language cannot accommodate them, perhaps because there is inadequate theoretical understanding of the domain in which the system operates. In any case, the operators must have some knowledge allowing them to pick up where the machine leaves off.

In DECSUP (section 8.2) the algorithmic and manual operations are interchangeable in the sense that the decision-maker can develop part of a decision via manual methods and then continue with the algorithm, or vice-versa.
Reliability

Transmission of data can fail in many ways. Knowing the ways that are most likely can focus efforts on improving design, or help choosing among competing designs. Detailed quantitative modelling of communicational reliability might highlight such vulnerabilities.

In this study the integrity of transmitted data is suggested to be maintained through the use of either link encryption or end-to-end encryption. However, end-to-end encryption is believed to provide a higher level of reliability in a network environment. Reliability of such environments may also be accomplished through backup systems and alternative communication paths.

10.2. Conclusions and prospect

Conclusions

According to most prescriptive schemes, good decision making involves the following steps (Chapter 3):

1) Identify all possible courses of action (including, perhaps, inaction).

2) Evaluate the attractiveness (or aversiveness) of the consequences that may arise if each each course of action is adopted.

3) Assess the likelihood of each consequence actually happening (should each action be taken).

4) Integrate all these considerations, using a defensible (i.e., rational) decision rule to select the best (i.e., optimal) action.

The discussion in the previous section has shown difficulties in each of these steps at a military headquarters. However, it can be concluded from the results of the inquiry presented in Chapter 9 that supporting decision-making by DECSUP through adherence to the concepts of
distribution introduced in this study in relation to the aforementioned steps has produced improvement in decision-making effectiveness at a military headquarters.

**Prospect**

Individuals in organizations must continually learn about their environment and about themselves if the organization is to succeed. Organizational structure can facilitate or impede learning and the distribution of information. Our work may have effects on learning processes inasmuch as it relates to methods by which experience is accumulated and disseminated. Learning and dissemination are inevitably secondary to the organization's objectives. They cannot, however, be ignored.

In a distributed decision-making context, the primary purpose of acquiring information is to make better decisions. This naturally raises the question of the value of the underlying information. For example, while the knowledge of the winning lottery numbers in last week's and next week's drawing have the same information content in the Shannon sense, they have very different values.

The study of information value and structure in a military organization should lead to a cost-benefit analysis of 'who should know what?'. However, in general we are far from being able to answer such a query systematically and in a unified manner. Several difficulties are involved. First, our understanding of the dynamic of information flow in military headquarters is meager. One purpose of this study was to convey a sense of both the difficulty and the richness of the subject. Much more insight, for example, on the matter of 'different grades of communication' is needed before more sophisticated algorithms of distributed decision-making support can be realized. In this sense, much of the current work can be thought of as preparatory and basic.

Secondly, the question of value of information implies the ability to determine the effectiveness of a particular decision with or without the particular piece of information. In view of the difficulties of solving
decision problems in dynamic information flow structure, one should try to devise methods of ordering information flows without having to solve the decision problem involved.

Since optimal solutions for decision-making are difficult to obtain, one is often forced to ask a less ambitious question, such as what interesting properties of distributed decision support can we discover if we restrict a priori the range of admissible strategies?

In one sense, in restricting the admissible strategies to (e.g., coordination with no communication), one has assumed away the possibility of choosing alternative decisions related to other strategies. Even if an optimum can be achieved with the selected strategy, there is usually no guarantee that efficiency cannot significantly improve if another strategy is used.
Bibliography


Bibliography


Bibliography


Bibliography


Bibliography


Bibliography


Bibliography


How DUELING works

DUELING works in two ways: It parses the rule base and then checks it for errors. Since we wanted to concentrate on the error-checking feature, we decided to forego a bulletproof routine to allow input of the name of file to check.

The program uses the read/2 predicate to read rules from the input file. Read/2 denotes the predicate read, which is called with two arguments. The read/2 predicate has the advantage of immediately grabbing an entire rule and ignoring embedded comments. The program stores rules with a numerical tag that helps to distinguish them and gives the program the option of using the for/3 predicate (a passable implementation of a FOR loop in PROLOG) to cycle through the rules.

The predicates dissect/2, dissect1/2, and dissect2/2 work together to pick apart each rule (see listing 1). First, dissect/2 checks to see that its first parameter is a structure (i.e. not an atom). The uniop operator (=..) then transforms the parameter into a list. For example, the term con(A,B,C) :- ant(D,E,F) becomes [.., con(A,B,C), ant(D,E,F)]. The dissect1/1 predicate then passes this list as an input parameter to the dissect2/2 predicate. The dissect2/2 predicate then recursively 'walks' down the list and subjects each member to a similar procedure that call dissect1/2 for the head of the input list, and dissect2/2 for the tail. In the end, the above example looks like

[.., [con, A, B, C], [ant, D, E, F]].

One weakness of this routine is that it cannot distinguish between an atom and a name of a zero-arity predicate, so that a term like con(A, B, C) :- ant. dissects into [.., [con, A, B, C], ant]. For the sake of consistency, the predicate ant should be enclosed in list brackets in this example. On the other hand, a term like open(X, Y, r), where r is an atom, dissects correctly into [open, X, Y, r].

The program compensates for the weakness of the dissection predicates by calling the massage_null_arity_predicates/2 predicate. This code assumes that if the first element of the input parameter is an operator,
then the remaining elements of the list should be processed; otherwise
the input parameter is left unchanged. This predicate correctly changes [\:-, [con, A, B, C], ant] into [\:-, [con, A, B, C], [ant]] but leaves [open, X, Y, r] alone.

Another important step changes all rules of the form X. into X :- true.
(which is what the first form implies), giving each rule a consequent and
an antecedent part. Although traditional logic statements position
antecedents to the left and the consequent to the right of the implication
sign, as in this example:

Antecedent  \rightarrow  Consequent

the positioning is just the reverse in the well written PROLOG clause, like
so:
Consequent :- Antecedent(s).

Once the rule base is completely normalized, the error checking can
begin.

Listing 1: Definitions of DUELING predicates that pick apart PROLOG
rules. dissect/2 takes a structure and returns a list. massage_null arity predicates/2 helps normalize those lists.

/* dissect/2 */

dissect(H,Z) :-
    structure(H),
    univ(H,D),
    !,
    dissect2(D,Z).

dissect1(H,Z) :-
    univ(H,D),
    [H] \== D,
    !,
    dissect2(D,Z).

dissect1(H,H).

dissect2([],[]) :- !.

dissect2([H|T],Y) :-
    dissect1(H,Z0),
    dissect2(T,Z1),
$Y = [Z_0|Z_1],!.$

```prolog
message_null arity predicates(In,Out) :-
   In = [Operator,Operand1,Operand2],
   current_op(_,_,Operator),
   deatomize(Operand1,NewOperand1),
   message_null_arity_predicates(Operand2,NewOperand2),
   Out = [Operator,NewOperand1,NewOperand2],
   !.
message_null_arity_predicates(In,In) :-
   !.
   deatomize(In,[In]) :-
      not(list(In)),
      not(structure(In)),
      atomic(In),
   !.
   deatomize(X,X).
```

What DUELING does

There are a number of interesting and subtle ways that a rule base can go wrong. The prototype presented here does, however, attempt to illustrate some of the more common problems and ways in which one can detect them.

Perhaps the easiest check one can perform on a PROLOG rule base is for duplication. Duplicate rules invite erratic program behaviour. If DUELING finds two rules that unify, than a duplicate has been found.

The next test performed by DUELING looks for what is called 'irregularities'. When the find_irregular_rule_predicate is called, PROLOG looks for instances of irregular/2 in its database. An example of an irregular relationship is:

```
irregular(male(X),female(X)).
```

which simply notes that X cannot be both male and female. Currently, the only irregularities that DUELING considers are those that involve contradiction, but there is certainly room for expansion in this direction.
The user must separately compile these irregular/2 clauses and then make them part of either the DUELING source code or a file that is consulted in conjunction with DUELING. In preparation for further error checking, DUELING combs through the rule base and picks out all the predicates on the antecedent side of each rule.

Once the program finds antecedents for each rule, DUELING can perform additional error detection. A less serious error in rule bases is the presence of what is called 'orphan rules'. These are rules that will never fire, because the consequent of the rule never appears as an antecedent in any other rule. In some cases, these orphans are nothing more than utility predicates that have no direct bearing on the code. Most of the time, however, these orphans do nothing but take up space and are errors waiting to happen as long as they are present.

Using PROLOG's powerful findall/3 predicate, DUELING finds such rules quickly. It's important to note that every PROLOG program has at least one orphan, specifically the name of the predicate that got the whole thing started.

Analogous to orphan rules are 'unfireable' rules. These are rules that will never fire because at least one of the consequent to the rule has no way of being satisfied. For example, the rule

\[ A \leftarrow X, Y, Z \]

will always fail if no clauses exist in the database for X, Y, or Z. We have found this error to be one of the most difficult to trap without close examination of the rule base text. The program cannot differentiate between user-written predicates that are unfireable and standard predicates that, by definition, have no defining clauses in the rule base.

To remedy this weakness and cut down on the volume of DUELING's output, we used the std_predicate/1 clause, which checks to see whether a predicate is a standard predicate before DUELING judges a rule to be unfireable.
Finally, the last error that DUELING tries to identify deals with subsumption. Simply stated, two rules subsume one another if they both have the same consequence and one rule's antecedents are a subset of the other rule's antecedents. This relationship is illustrated in the following example:

\[
\text{grandparent}(X, Y) : - \\
\quad \text{parent}(X, Z), \text{parent}(Z, Y).
\]

\[
\text{grandparent}(X, Y) : - \\
\quad \text{parent}(X, Z), \text{parent}(Z, Y), \text{has}(\text{gray\_hair}, X).
\]

For the two shown rules, it is immaterial whether \(X\) has gray hair, as long as \(X\) is the parent of \(Z\), and \(Z\), in turn, is the parent of \(Y\). This rule is sure to fire at least once, and erroneously twice if \text{has}(\text{gray\_hair}, X)\) is true.

The Future

There is tremendous room for expanding the scope of this prototype. Duetring's most obvious limitation is its ability only to deal with PROLOG rules. Despite this constraint, one could write extensions to allow for the modularisation of a program among several PROLOG source files and still keep Dueling from identifying spurious errors.

Other errors that may run across include detection of infinite recursion, detection of synonyms and aliases, and comparison to a set of legal values.

Despite these limitations, Dueling is still a useful program, primarily because of the errors it is able to detect, and the fact that it helps to prove the hypothesis of this work. With more tools like Dueling, knowledge engineers will have greater control over the rule bases they create; eventually, the tools will provide a means to reliably verify and validate rule bases.
SUMMARY

Background
Before the study at hand has started there were some temporary objectives. The first objective involved the provision of the command level of a military headquarters with all the information needed for decision-making, with the help of an information system. The second objective consisted of an investigation of the possibilities of integrating such a system into the decision-making process, together with a demonstration of this integration using several examples. This provided the basic requirements for the development of a decision support system (DSS).

During the course of analysis of these requirements it was found that the decisions devolving upon the headquarters investigated were mainly of a strategic nature, and that not all decisions were made centrally. Therefore the goal was redefined as the development of a distributed decision support system (DDSS).

Overview
This study of distributed decision-making and the DDSS development work is divided into the areas of environment and characteristics (Chapters 1, 2, and 3), models, requirements, and algorithms (Chapters 4, 5, and 6), decision-makers and users (Chapter 3 partly and Chapter 7), and system implementation and evaluation (Chapters 8 and 9). We will begin by describing the environment, and then proceed with the other areas.

Environment and characteristics
The environment is a military headquarters. However, during the course of the project it became apparent that the prototype DDSS we developed could be used in most hierarchical organizations.

Decisions are taken by the commander based on briefings. These decisions are part of a functional area. Some examples of functional areas are crisis management, logistics, and NBC-defense. Information touching upon the functional areas is supplied through external communications channels. The sources of this information are the respective defense ministries and other headquarters.
Summary

How are decisions reached in a military headquarters? Information on a certain subject is collected and assessed; plans are then made, on the basis of which a decision is taken. This decision is converted into one or more military tasks, and passed on to the appropriate units for execution. The results of this execution are collected, evaluated, and the "Command and Control cycle" continues with planning.

This cycle operates not only at the strategic level, but also at the tactical and the operational level.

The reason for this detailed look at this cycle is the description of the individual tasks at each phase of this cycle for each headquarters. These descriptions are assembled in the so-called "Key Task List". Thus, for the decision phase there is an exact description of the decisions relevant for a certain headquarters. These are the so-called "Decision Key Tasks". The "Key Tasks" or "essential command functions" are collected into functional areas, and clearly belong to a certain phase of the cycle.

Models, requirements, and algorithms
On the basis of Key Tasks a model of a military headquarters is constructed that is based on logic modelling and that has components like Finite automata (FA) and a directed graph.

There are many kinds of information-flow interlinkages between Key Tasks and thereby between functional areas. This makes it very difficult to keep track of which information is needed where and which influence it thereby has. Our model serves as the basis for the DDSS we developed, which helps to get over these difficulties.

The Decision Key Tasks are performed more or less with the aid of other information systems. For a DDSS to be effective, it must have access to the findings and conclusions reached using other systems. If this information flow is lacking, any decision is going to be very risky, regardless of whether it is taken with the aid of a DDSS or not. But this is exactly what a commander does not want!

So, the DDSS must be able to store, organize, and access information, and process it graphically.
Summary

As a consequence a number of strategies using varying grades of communation to reduce uncertainty in each commander's decision-making are presented.

In order for a DDSS to be effective as more than just an information retrieval system, an algorithm is created with the help of the rules of non-monotonic logic. This algorithm can analyze events to make predictions as to the consequences of decisions and is described as follows.

"Events" are interpreted as incoming messages, decisions made by other commanders, own decisions, or milestones - important occurrences. There are initial conditions, such as a state of crisis. Something happens, like a bombing attack. This event creates new conditions. Consequences result. A new event happens, and so forth.

Applying this pattern to a military headquarters, it quickly becomes extremely difficult to track of which event resulted in which consequences and which conditions resulted in which event. The structure of events and consequences depends on their chronological sequence, and are described by a partial order.

Based on non-monotonic logic, an algorithm is developed which calculates the consequences of events. An event has a series of consequences. Some of these consequences are necessary or inescapable, that is, they represent the lower limit of possible consequences. Others are merely conceivable, and in sum represent the upper limit of all possible consequences. Taking past events as a basis, the algorithm calculates new upper and lower limits for the consequences of new events.

Decision-makers and users

We pay special attention to the potential users of a DDSS. In developing the user interface, we make use of findings by cognitive psychologists to compensate for the weaknesses of human information processing in the decision-making process.

The most important element in the decision-making process is the decision maker himself. The most important preparation for reaching a
Summary
decision is the processing of incoming information by a human. It follows
that we must study the way in which humans process information and
find any areas of weakness, so that these areas can be compensated by a
decision support system. There are essentially five limitations in human
information-processing that can be compensated by a DSS.

One example of these limitations is the inability to predict the spatial and
chronological sequence of events in physical processes. For example, take
the traces of two aircraft on a radar screen. Tests have shown that
humans are unable to predict either the probable point of intersection of
the two flight paths, or the time of intersection. For a computer, on the
other hand, this is a simple task.

From these limitations, and from many discussions with commanders the
requirements for the user interface of a DDSS are described as follows.

Staff do not necessarily base decisions on facts, but instead on how they
apprehend the facts. This means there is a need for the sophisticated
graphic display of facts.

A person faced with making a decision has a certain objective in mind.
His decision-making is goal-oriented. The definition of objectives for each
phase in the decision-making process gives us a structure which
corresponds to human decision-making.

Information is mentally processed and understood by organizing it
among known categories. Therefore, we must prevent information from
getting "stuck in the wrong pigeonhole". We must replace the subjective
categorization of information with an objective categorization.

Examples of a system/user interface are presented, which incorporate
these requirements.

Implementation and evaluation
In our description of the implementation of the system we briefly discuss
the basic principles of data integrity and the development method we
use, and give an idea of the time investment involved.
As already said the environment for the system is a military headquarters and it is linked by a wide area network to various other headquarters and backup headquarters. At headquarters there are information systems with several different security levels. However, for a DDSS to function, it is absolutely necessary for it to have access to data in other systems. This gives an example of the problems with data integrity. A possible solution is to use certain security filters for access from the DDSS to other systems.

For the implementation of the system we chose an evolutionary approach. This begins with a so-called "throwaway prototype", developed together with the future system user. When this prototype has reached a stage where a decision can be reached for or against further development into an operational system, it is thrown away. It has done its job. Development is started on a new prototype, avoiding wherever possible the mistakes and weaknesses of the first. Improvement of this prototype is an iterative process, in which the user should be heavily involved. This new prototype becomes the operational system.

An example of an operational system is shown within the NATO Alert System (Figure 6-8). The decision involved here is the determination of the necessity for the declaration or cancellation of a certain alarm status for all NATO forces in Europe. Each of the boxes in the example contains an abbreviated description of a Key Task. For the "maintain status" phase the following tasks must be carried out in a military headquarters: Determination of the status of enemy and allied forces. Assembly of recommendations for alarm status from subordinate headquarters. Determination of the current status of all allied nations and status of international negotiations. After assessing enemy and allied forces, and taking into account other status information, a recommendation must be worked out for the declaration or cancellation of a certain alarm status. The decision for or against the recommendation, in this case a recommendation from Supreme allied Commander Europe (SACEUR) to the Defense Planing Council (DPC), must be taken by SACEUR itself. The consequences of the acceptance or rejection of this recommendation can be very far-ranging indeed.

To present a decision situation a description format based on broad categories of analysis that are common to most decisions is defined. This
description format is a revised and modified version of the so called "Summary Tabulation of Aiding Requirements" or STAR Table. An extended example using this format is given for the "on-station search" situation in an Air Anti Submarine Warfare mission.

A screen layout also using the STAR Table format is shown for another example. The decision situation is: "Decide need for declaration or cancellation of a certain alert status". The "Situational objectives" might be a satisfactory response to an inquiry from the Defense Planning Council. The "Input" shows available reports, maps, and recommendations. "Output" takes the form of decisions, guidelines, or orders. If desired, the user can also enter a hypothetical decision and examine the potential consequences.

The upper right-hand corner is for navigation through the results of the individual phases of the Command and Control cycle provided from other information systems.

The yardsticks by which the DDSS is evaluated are system efficiency and effectiveness. The numerous results of the evaluation are discussed in detail and several important findings made during the evaluation phase are presented. The results bespeak the high effectiveness of the system under observation. Regarding efficiency, the objection can be made that since the users can be ordered to use the system, this criterion is irrelevant. In such cases, a more representative criterion for system evaluation is user satisfaction.

Finally, we want to point out that no new technology is required to integrate existing systems and implement a DDSS. The technology is available now - it need only be used.
CURRICULUM VITAE

Name : Theodor W. Schlickmann
Nationality : German
Date and place of birth : 8.2.1948 in Gelsenkirchen

Education : Staatliche Ingenieurschule für
Maschinenwesen in Jülich (1969 - 1972),
Bachelor degree in nuclear-chemistry;

Technische Universität Karlsruhe
(1972 - 1975),
Master degree in mathematics;

Experience : General Electric Informations Service,
Frankfurt (1976 - 1979);

Aristo Graphic Systeme, Hamburg
(1979 - 1981);

Independent consultant for various companies (1982 - 1988);

SHAPE Technical Centre, Den Haag
(1988 - 1991);

Hanseatische Unternehmensberatung,
<table>
<thead>
<tr>
<th>Index</th>
<th>Page Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>functional areas</td>
<td>137</td>
</tr>
<tr>
<td>future possibilities</td>
<td>40</td>
</tr>
<tr>
<td>goal</td>
<td>17, 87</td>
</tr>
<tr>
<td>goal decomposition</td>
<td>165</td>
</tr>
<tr>
<td>goal-driven deduction</td>
<td>16</td>
</tr>
<tr>
<td>heuristic judgments</td>
<td>38</td>
</tr>
<tr>
<td>hierarchical system</td>
<td>73</td>
</tr>
<tr>
<td>higher-level commanders</td>
<td>161</td>
</tr>
<tr>
<td>homo ludens</td>
<td>41</td>
</tr>
<tr>
<td>human behaviour</td>
<td>117</td>
</tr>
<tr>
<td>human factors</td>
<td>31</td>
</tr>
<tr>
<td>human information-processing</td>
<td>33</td>
</tr>
<tr>
<td>human interactive processor</td>
<td>113</td>
</tr>
<tr>
<td>human limitations</td>
<td>26</td>
</tr>
<tr>
<td>human operators</td>
<td>167</td>
</tr>
<tr>
<td>human-computer interaction</td>
<td>105</td>
</tr>
<tr>
<td>human-factor</td>
<td>103</td>
</tr>
<tr>
<td>human-factors research</td>
<td>32</td>
</tr>
<tr>
<td>hybrid model</td>
<td>81</td>
</tr>
<tr>
<td>implementation</td>
<td>26</td>
</tr>
<tr>
<td>incentive systems</td>
<td>1</td>
</tr>
<tr>
<td>induction relation</td>
<td>73, 81</td>
</tr>
<tr>
<td>information</td>
<td>27</td>
</tr>
<tr>
<td>Information environment</td>
<td>129</td>
</tr>
<tr>
<td>Information sources</td>
<td>28</td>
</tr>
<tr>
<td>information systems</td>
<td>27</td>
</tr>
<tr>
<td>information value</td>
<td>169</td>
</tr>
<tr>
<td>initial</td>
<td>56</td>
</tr>
<tr>
<td>Initial conditions</td>
<td>88</td>
</tr>
<tr>
<td>interacting commanders</td>
<td>52</td>
</tr>
<tr>
<td>Interface Units</td>
<td>116</td>
</tr>
<tr>
<td>Intermediate analysis</td>
<td>130</td>
</tr>
<tr>
<td>Interpretation</td>
<td>65</td>
</tr>
<tr>
<td>irregularly structured system</td>
<td>75</td>
</tr>
<tr>
<td>ISO</td>
<td>143</td>
</tr>
<tr>
<td>iterative prototyping</td>
<td>47</td>
</tr>
<tr>
<td>iterative techniques</td>
<td>53</td>
</tr>
<tr>
<td>Key Tasks</td>
<td>5</td>
</tr>
<tr>
<td>lattices</td>
<td>13, 14, 54</td>
</tr>
<tr>
<td>limitations</td>
<td>34</td>
</tr>
<tr>
<td>limited future period</td>
<td>77</td>
</tr>
<tr>
<td>limited-horizon coordination</td>
<td>79</td>
</tr>
<tr>
<td>link encryption</td>
<td>143</td>
</tr>
<tr>
<td>logic modelling</td>
<td>21, 164</td>
</tr>
<tr>
<td>logic programming</td>
<td>15, 20, 87</td>
</tr>
<tr>
<td>logistics</td>
<td>23</td>
</tr>
<tr>
<td>Macintosh</td>
<td>49, 127</td>
</tr>
<tr>
<td>maintenance</td>
<td>7</td>
</tr>
<tr>
<td>management information systems</td>
<td>28</td>
</tr>
<tr>
<td>manual operation</td>
<td>105</td>
</tr>
<tr>
<td>mathematical programming</td>
<td>20</td>
</tr>
<tr>
<td>MAU assessment</td>
<td>155</td>
</tr>
<tr>
<td>measurements</td>
<td>101</td>
</tr>
<tr>
<td>measures</td>
<td>155</td>
</tr>
<tr>
<td>meta-interpreter</td>
<td>17</td>
</tr>
<tr>
<td>metasystem</td>
<td>109</td>
</tr>
<tr>
<td>military headquarters</td>
<td>1</td>
</tr>
<tr>
<td>military users</td>
<td>40</td>
</tr>
<tr>
<td>min-max strategy</td>
<td>80</td>
</tr>
<tr>
<td>model-components</td>
<td>43, 69</td>
</tr>
<tr>
<td>modelbase</td>
<td>69</td>
</tr>
<tr>
<td>modelling</td>
<td>39</td>
</tr>
<tr>
<td>modularity</td>
<td>19</td>
</tr>
<tr>
<td>multi-level secure</td>
<td>142</td>
</tr>
<tr>
<td>NATO-CCIS</td>
<td>121, 137</td>
</tr>
<tr>
<td>natural-language processing</td>
<td>67</td>
</tr>
<tr>
<td>Necessary(e)</td>
<td>89</td>
</tr>
<tr>
<td>negotiation</td>
<td>22</td>
</tr>
<tr>
<td>neighbour relationship</td>
<td>54, 72</td>
</tr>
<tr>
<td>no communication</td>
<td>77, 79</td>
</tr>
<tr>
<td>non-chronological backtracking</td>
<td>90</td>
</tr>
<tr>
<td>non-monotonic logic</td>
<td>102</td>
</tr>
<tr>
<td>non-monotonic reasoning</td>
<td>89</td>
</tr>
<tr>
<td>normatively acting commander</td>
<td>34</td>
</tr>
<tr>
<td>objective functions</td>
<td>21</td>
</tr>
<tr>
<td>objectives</td>
<td>76</td>
</tr>
<tr>
<td>on-station search</td>
<td>134</td>
</tr>
<tr>
<td>operational requirements</td>
<td>3, 120</td>
</tr>
<tr>
<td>Organizational structure</td>
<td>169</td>
</tr>
<tr>
<td>organizational structures</td>
<td>71</td>
</tr>
<tr>
<td>OSI</td>
<td>143</td>
</tr>
<tr>
<td>outputs</td>
<td>76</td>
</tr>
<tr>
<td>partial order</td>
<td>12</td>
</tr>
<tr>
<td>partially ordered events</td>
<td>87</td>
</tr>
<tr>
<td>planning</td>
<td>8, 166</td>
</tr>
<tr>
<td>politico-military situation</td>
<td>142</td>
</tr>
<tr>
<td>Index</td>
<td>Page</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>possible consequences</td>
<td>76</td>
</tr>
<tr>
<td>Possible(e)</td>
<td>89</td>
</tr>
<tr>
<td>preconditions</td>
<td>93</td>
</tr>
<tr>
<td>predicting consequences</td>
<td>86</td>
</tr>
<tr>
<td>preference orderings</td>
<td>22</td>
</tr>
<tr>
<td>procedural style</td>
<td>17</td>
</tr>
<tr>
<td>PROLOG</td>
<td>15, 18, 67, 85, 87</td>
</tr>
<tr>
<td>protocols</td>
<td>143</td>
</tr>
<tr>
<td>prototype</td>
<td>45</td>
</tr>
<tr>
<td>prototyping</td>
<td>44</td>
</tr>
<tr>
<td>questionnaires</td>
<td>159</td>
</tr>
<tr>
<td>relation</td>
<td>12</td>
</tr>
<tr>
<td>required judgments</td>
<td>130</td>
</tr>
<tr>
<td>requirements analysis</td>
<td>59</td>
</tr>
<tr>
<td>role of logic</td>
<td>19</td>
</tr>
<tr>
<td>rules</td>
<td>88</td>
</tr>
<tr>
<td>scenario</td>
<td>125</td>
</tr>
<tr>
<td>security</td>
<td>142</td>
</tr>
<tr>
<td>selection criteria</td>
<td>128</td>
</tr>
<tr>
<td>semantics of logic programs</td>
<td>16</td>
</tr>
<tr>
<td>separation</td>
<td>84</td>
</tr>
<tr>
<td>single instance</td>
<td>128</td>
</tr>
<tr>
<td>singly connected</td>
<td>72</td>
</tr>
<tr>
<td>situational objectives</td>
<td>127</td>
</tr>
<tr>
<td>software structure</td>
<td>122</td>
</tr>
<tr>
<td>sonobuys</td>
<td>133</td>
</tr>
<tr>
<td>space logistics</td>
<td>23</td>
</tr>
<tr>
<td>space station</td>
<td>23</td>
</tr>
<tr>
<td>Specification</td>
<td>65</td>
</tr>
<tr>
<td>spreadsheet package</td>
<td>17</td>
</tr>
<tr>
<td>standard actions</td>
<td>5</td>
</tr>
<tr>
<td>STAR Table</td>
<td>131</td>
</tr>
<tr>
<td>state of a computation</td>
<td>16</td>
</tr>
<tr>
<td>state transitions</td>
<td>57</td>
</tr>
<tr>
<td>statements</td>
<td>87</td>
</tr>
<tr>
<td>static situation</td>
<td>78</td>
</tr>
<tr>
<td>static strategy</td>
<td>82</td>
</tr>
<tr>
<td>stochastic approaches</td>
<td>20</td>
</tr>
<tr>
<td>strategic decisions</td>
<td>4</td>
</tr>
<tr>
<td>Strong(e)</td>
<td>93</td>
</tr>
<tr>
<td>structured programming</td>
<td>19</td>
</tr>
<tr>
<td>sub-tasks</td>
<td>87</td>
</tr>
<tr>
<td>super-strong(e)</td>
<td>94</td>
</tr>
<tr>
<td>super-weak(e)</td>
<td>94</td>
</tr>
<tr>
<td>symbolization</td>
<td>59, 62</td>
</tr>
<tr>
<td>tactical decisions</td>
<td>4</td>
</tr>
<tr>
<td>task dynamics</td>
<td>127</td>
</tr>
<tr>
<td>task-interactive processors</td>
<td>113</td>
</tr>
<tr>
<td>TEMPEST</td>
<td>146</td>
</tr>
<tr>
<td>term</td>
<td>15</td>
</tr>
<tr>
<td>terrorist bombing</td>
<td>2</td>
</tr>
<tr>
<td>threats to C2</td>
<td>5</td>
</tr>
<tr>
<td>time pressure</td>
<td>4</td>
</tr>
<tr>
<td>timeliness</td>
<td>153</td>
</tr>
<tr>
<td>totally ordered extensions</td>
<td>88, 96</td>
</tr>
<tr>
<td>transition function</td>
<td>56</td>
</tr>
<tr>
<td>transition functions</td>
<td>58</td>
</tr>
<tr>
<td>transition issues</td>
<td>46</td>
</tr>
<tr>
<td>truth maintenance</td>
<td>90</td>
</tr>
<tr>
<td>turing machine</td>
<td>16</td>
</tr>
<tr>
<td>underlying process</td>
<td>128</td>
</tr>
<tr>
<td>unfolding</td>
<td>128</td>
</tr>
<tr>
<td>UNIX</td>
<td>119</td>
</tr>
<tr>
<td>user dimensions</td>
<td>117</td>
</tr>
<tr>
<td>user interface</td>
<td>125</td>
</tr>
<tr>
<td>user/system interfaces</td>
<td>70</td>
</tr>
<tr>
<td>Value of information</td>
<td>152</td>
</tr>
<tr>
<td>visual interactive modelling</td>
<td>43</td>
</tr>
<tr>
<td>visualizing</td>
<td>38</td>
</tr>
<tr>
<td>voting procedure</td>
<td>3</td>
</tr>
<tr>
<td>vulnerabilities of C2</td>
<td>5</td>
</tr>
<tr>
<td>Weak(e)</td>
<td>93</td>
</tr>
<tr>
<td>workload description</td>
<td>126</td>
</tr>
<tr>
<td>worst case</td>
<td>53</td>
</tr>
</tbody>
</table>