A Generic Approach to the Development of Tactical Decision Aids

Martijn Neef
TNO Physics and Electronics Laboratory
PO Box 96864
2509 JG The Hague
THE NETHERLANDS

Guido te Brake
TNO Human Factors
PO Box 23
3769 ZG Soesterberg
THE NETHERLANDS

1.0 INTRODUCTION

Military tactics require efficient and effective decision making in unclear, complex and dynamic environments. Tactical decision-makers need to make the right choices based on large amounts of data from various sources with varying reliability and often under severe time-pressure. Therefore decision makers usually have a set of Tactical Decision Aids (TDA's) available to them. Even though TDA's can be found anywhere, they often fail to live up to expectations in terms of quality of support. Many efforts to design effective decision aids for command and control domains have been unsuccessful due to three main reasons (Essens et al., 1994): a shallow understanding of the C2 decision making process; a technology driven design process; and a design effort based on not well-analyzed decision making problems. Additionally, since decision aids are very problem-specific (i.e. they require a large amount of domain knowledge in order to provide support), they are costly to design. This paper describes an effort undertaken by TNO to develop a generic design strategy that addresses both the cognitive issues as well as the technical issues involved with TDA's. We base our approach around the COADE framework and demonstrate it in a small scale case study based around the task of sensor management.

2.0 A GENERIC APPROACH TO TDA DEVELOPMENT

The subject of this project to develop a generic methodology to the development of TDA's. The key issue here is that we are not focussing on the development of a single system from scratch, but that we are rather engineering a family of systems: decision support systems within a tactical environment. With respect to reusability and flexibility, we can take advantage of common properties that make up family of systems and use these to formulate 'generic' building blocks for our framework. These building blocks will ultimately be specialized to fit the specific requirements of a specific TDA. A generic methodology, in our view, entails two major elements: a design methodology and a generic architectural framework. A design methodology can best be described as a well-structured and detailed process plan on how to arrive at a system that meets all applicable requirements. An architectural framework is a technical reference model, from which a broad range of different systems can be developed. It displays system characteristics that are present throughout the whole family of systems, but allows for easy problem-specific instantiation of functionality. Usually it includes a set of basic building blocks and a description on how these fit together.

The reason to devise such a generic approach is fairly pragmatic. With increasing operational complexity of military operations and decreasing development budgets, there is an evident need for better decision support with lower development costs. By providing a well laid-out process plan, the development process can be sped up and pitfalls can be avoided more easily. A related reason is to promote reusability in TDA design.

Paper presented at the RTO SCI Symposium on "Critical Design Issues for the Human-Machine Interface", held in Prague, Czech Republic, 19-21 May 2003, and published in RTO-MP-112.

Most decision support systems (and information systems in general) contain similar functionality on various levels of information abstraction. This property opens up opportunities for reusable components, or at the least, reusable design concepts. Another important aim is to stress the importance of the cognitive aspects of decision making in TDA design. Decision aids can only be successful if they harmonize seamlessly with user needs, so a user-centered design methodology is invaluable.

The generic approach we are presenting in this paper is based around the COADE framework (Essens et al, 1994), and complemented with a generic, agent-oriented architecture concept.

3.0 COADE: ESTABLISING FUNCTIONAL REQUIREMENTS

Establishing functional requirements for tactical decision aids requires analyzing task, personal and environmental constraints. This section reintroduces a previously developed framework, COADE, for the design of decision support systems, which provides a skeleton for models that provide requirements for the decision aid and provides essential guidelines for designing tactical decision aids.

3.1 COADE

The COADE framework was developed within NATO Defence Work Group panel 8 – RSG.19. COADE is a framework with a strong emphasis on cognitive factors for the development of DSS's for complex environments. The goal of COADE is to assist in the development of decision support based upon knowledge of the human role, capabilities, and the tasks to be performed. COADE provides a framework for:

- The identification of cognitive requirements of the tasks (ANALYSE);
- The development of design requirements that address those cognitive requirements (DESIGN);
- The assessment of the quality of intermediate and final products or results in the developmental process (EVALUATE).

During ANALYSE, the goal is to identify the critical aspects of the system, which results in a specification of critical cognitive requirements on which design decisions should be based. The activity set ANALYSE consists of two phases:

1) *Task and performance analysis*. In this phase, task characteristics, person characteristics and the environmental characteristics with respect to a task are determined and evaluated. Task analysis yields a description of the task under research and performance analysis assesses whether the task is performed according to quality requirements.
2) *Cognitive task analysis and cognitive performance analysis*. Cognitive task analysis produces a description of a task in cognitive terms. It describes how memory is involved, what knowledge, facts and procedures are involved; which strategies are used to solve problems, which goals are formulated explicitly and implicitly to achieve the objectives; what judgements need to be made and what decisions have to be taken. The question of how well people perform cognitively is assessed during cognitive performance analysis.

The purpose of DESIGN is to translate cognitive requirements into system design specifications. There is no one-to-one mapping between cognitive requirements and solutions. Prototypes support the fast turnaround of ideas and user feedback. When a computer-based solution is being considered a detailed specification of the interaction between the human and the computer must be developed. The design of an interface is a complex and subtle matter. The critical issue is to understand the goal structure of the user at this level of execution of the task and to provide adequate feedback in the dialogue. Development of a user interface follows an iterative cycle of analysis, design and evaluation.

In EVALUATE, the developer steps aside for a moment and verifies the results of the analysis and design activities in a more or less formal way. The evaluation of a newly build system is often carried out as a final phase of the development process. COADE embeds the EVALUATE activity within the ANALYSE and DESIGN activity sets. Whenever intermediate products are delivered, they are to be evaluated immediately. Results that are evaluated are the behavior model, cognitive requirements, solutions, and interface designs. Figure 1 visualises the COADE development process.

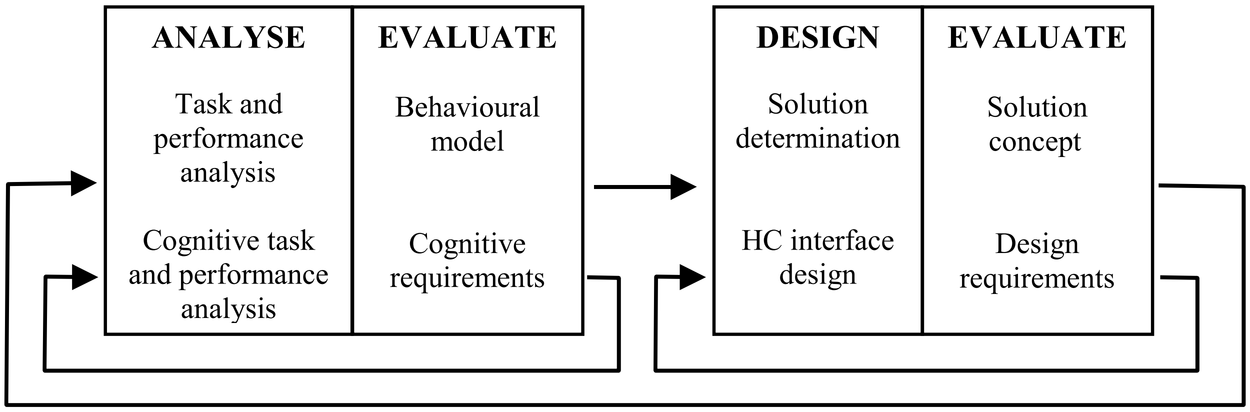

Figure 1: Brief Summary of the COADE Framework (Adapted from Essens et al. (1994)).

3.2 Basic Stages of COADE

The COADE framework can be summarized in five main steps:

1) Determine the tasks that the user has to perform using the structure the C2 process model provides with constraints on time and requirements for the required accuracy for these tasks.
2) Elicit decision-making principles from domain experts using decision-making models. Important topics are priorities, used heuristics and guidelines, skills and experience.
3) Examine cognitive tasks as established in step 3 for possible mental overload using a model of human cognitive capacities.
4) Determine which type of support is required or useful. What issues apply with respect to time pressure and complexity and task set management?
5) Design a user interface using the HCI guidelines. Add support features and provide required information and interaction possibilities.

We will elaborate on these steps in the following sections.

Step 1: Determining the Task using a C2 Model

Figure 2 shows a C2 process model that is very much appropriate within the context of TDA design (Passenier and van Delft, 1995). This generic C2 process model contains four functions:

1) *Situation Awareness (SA):* Building and maintaining the actual situational overview using (pre-processed) sensor data and additional sources of information.
2) *Threat Assessment (TA):* Interpreting the current situation from a tactical point of view.
3) *Decision-making (DM):* Planning of missions and counter-actions.
4) *Direction and Control (DC):* Deployment of actions.

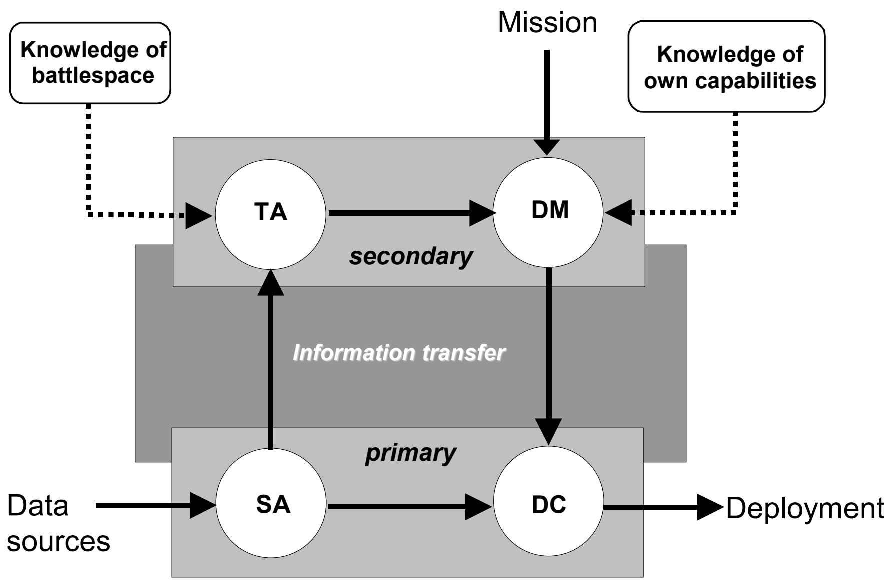

Figure 2: C2 Process Model with Two Levels of Information Processing (Passenier and van Delft, 1995).

The four generic C2 functions are arranged at two levels. The SA function provides a view of the current situation. On a primary level, actions can be taken directly by the DC function, according to the skill-based decision-making of Rasmussen (1986). At a higher level, the TA function uses domain knowledge to form a correct interpretation of what is happening. Based on this more complex high-level interpretation, the DM function may plan appropriate actions. The primary and secondary processes are running in parallel, the first is fast and (near) real-time (for example, the process of firing the goalkeeper), the latter is a long-term process that forecasts and plans which makes it possible to anticipate future developments (such as tactical planning).

Step 2: Models for Decision-Making

Basically, models for decision making can be divided into the “classical” models that regard decision making as a single-event choice between pre-generated options and the “naturalistic” models that are more focussed on “real-world” environments. Humans in complex situations make decisions quite differently from what would be expected by theory and laboratory experiments. A number of problems make real-life decision-making complex, for example incomplete information, high stakes, multiple players, uncertain and dynamic environment, ill-defined goals and time stress. The “classical” optimal decision-making theory loses validity in domains that suffer from these problems. Because these domains generally contain an enormous number of interrelated variables and situations that must be decoded and interpreted, people apply heuristics, such as:

- Satisficing, select the first option that satisfies the nebulous “requirements” necessary rather than trying to find the best
- Use various methods to quickly rule out non-solutions (compatibility testing)
- Focus on negative information to minimize risk (and hence missing opportunities)
- Focus on part of the data
- Focus on relevant data
- Extensive use of mental imagery

A number of Naturalistic Decision Making (NDM) models have been developed based on these principles, like *Image Theory, Recognition-primed Decision Making* (Klein, 1989, 1993) and *Story Telling* (Pennington and Hastie, 1993). With NDM, the decision-making strategies of the operator can be modeled to make a TDA that is more intuitive, because the cognitive processes of the operator are supported by the system. This could lead to more effective systems, because it prevents performance-degrading effects like confirmation bias, cognitive lockup and the delaying of actions. We advocate the use of NDM principles in the ANALYSE activity set for the cognitive task analysis and the cognitive performance analysis.

Step 3: Taskload Models

Cognitive capability and limitations are a well-studied subject in psychology. Rasmussen (1986) describes three levels of information processing: *skill-based*, *rule-based* and *knowledge-based*. At the skill-based level, information is processed automatically resulting into actions that are hardly cognitively demanding. At the rule-based level, input information triggers routine solutions (i.e. procedures with rules of the type ‘if <event/state> then <actions>‘), an efficient problem solving strategy in terms of required cognitive capacities. At the knowledge-based level, solutions for complex or new situations are constructed using input information. This type of information processing can involve a heavy load on the limited capacity of working memory. Neerincx et al. (2000) have developed a model expresses the mental load of an operator in terms of time occupation, level of information processing and the number of task-set switches, and which can provide the required information for this step of the COADE process.

Step 4: Types of Supporting Users

An important issue to consider is the manner in which the TDA should aid the human operator with his tasks. The system can present relevant information, advise the human operator what a good action could be, or even take over tasks and replace the operator. Which support form is best suited depends on the type of decision making problem and the operational context. Generally, five different modes of support are distinguished (Chalmers and Burns, 1999; Endsley and Kiris, 1995):

1) *Silent/Manual*: The operator has total authority. The system is completely passive.
2) *Informative:* The operator has total authority. The system only gives advice, probably partly on request by the operator.

3) *Cooperative mode*: The system and the operator work together. Authority can be divided or shared. In the latter case, either the system or the operator has the power to overrule the other.
4) *Automatic*: The system has total authority, but the operator can influence its behavior and veto decisions.
5) *Independent*: The system is fully autonomous and has complete authority. It can operate as a black box without any form of user interface.

In every mode of support well constructed user interaction is essential. Therefore TDA design can only succeed if the cognitive processes of the operator are understood and supported in the right way. Principles of naturalistic decision making should be taken into account. The support must be intuitive and consistent, data must be available in time and presented in a way that makes sense to the operator. If the operator is provided with all the important data in a manner that matches his cognitive decision-making process, he or she generally is capable of making good decisions. Consequently, the focus of many TDA's should be on organizing and visualizing the required data, and less on making the actual decisions.

Step 5: Human Computer Interface Design

Interface design can be considered as a process consisting of two phases that provide a user interface specification (Neerincx et al. 2001). In the first phase, based on users' goals and information needs, the system's functions and information provision are specified (i.e. the task level of the user interface is established). In the second phase, the control of the functions and the presentation of the information is specified (i.e. the "look-and-feel" or the communication level of the user interface is established). The following design principles are ordered in the same hierarchy: the task and communication level. Human factors principles at the task level are user adaptation, goal conformance, information needs conformance, user's complement and use context. These principles reflect the HCI-part of the cognitive requirements yielded by ANALYSE and the determined solution of DESIGN: what are the tasks and how are they going to be supported. Human factor principles at the communication level are compatibility, consistency, context, structure and pattern, feedback, interaction load, support and flexibility and maintenance (Williges et al., 1987). User-interfaces should be developed in an iterative way. First, a basic interface should be build, which is evaluated by the users. With their criticism and suggestions a second version of the interface is build, and so on. This process, called system evolution, prevents late detection of major mistakes and provides user comments and suggestions at an early stage in the design process.

3.3 TDA Development using COADE

Many system development methods are used in practice. A well-known method is the waterfall model, a linear model that builds systems in one very thorough pass of the ANALYSE and DESIGN phase. A drawback of this method is that errors that are made in the beginning of the ANALYSE process are often discovered late in the process during the programming of the software. Especially for complex systems with unclear user or system requirements, prototyping is a better approach. The prototyping model is a system development method in which a prototype (an early approximation of a final system or product) is built, tested, and then reworked as necessary until an acceptable prototype is finally achieved from which the complete system or product can be developed. When future users experiment with prototypes, errors and mistakes in the analysis and design process will be discovered early, resulting in lower costs. Prototyping with COADE entails that the ANALYSE, DESIGN and EVALUATE activity sets are repeated in an iterative way. With each step more detail is added until all system and user requirements are met and thoroughly tested.

4.0 A GENERIC TDA ARCHITECTURE

The DESIGN phase of the COADE framework lacks guidelines on how to actually build the TDA, and given the aim of COADE, this would also be well beyond its scope. For this reason we have devised a generic TDA architecture that a) is compatible with the functional goals of COADE, b) promotes reusability and c) is generic enough to accommodate the design of a wide variety of TDA's. We describe the architecture and its application in the following sections.

4.1 An Agent-Oriented TDA Architecture

The overall aim of our TDA development approach is to come up with a flexible and modular generic blueprint of a decision support system from which specialized TDA's can be derived. To this end, we need to provide basic building blocks, tied together in a framework and accompanied by fairly concrete description how the development of a TDA would be carried out, including problem analysis, human factors issues and software engineering aspects. COADE provides most of the user-centered cognitive factors and subsequent functional recommendations, and the overall development sequence. The technical architecture should adhere to the philosophy of COADE, thus allowing for a cyclic development process. To this end we need an architecture that is flexible and modular (i.e. is easily extensible and modifiable whenever new functional requirements are set) and that complies with prototype based development (i.e. is easy to develop in order to verify design concepts). Because of these requirements we deem an agent-oriented approach favorable. Using an agent-oriented approach to describe an architecture does not imply that software agents actually need to be used: there are plenty of COTS component-based approaches available, that do not use agents but can be described in the same way we are using agent principles.

For our purposes, we will loosely make use of a multi-agent-architecture introduced by Sycara and Zeng (Sycara, 1996), which later developed into the RETSINA architecture (Reusable Environment for Task-Structured Intelligent Networked Agents). The RETSINA architecture (Decker and Sycara, 1997) is a layered system in which agents are introduced in a task-specific way: there are interface agents, task agents, middle and information agents. Each of these agents has multiple goals and their activities jointly define a complex search process. RETSINA stands out because of its focus on reusability: albeit the various types of agents perform vastly different tasks, each RETSINA agent has four reusable modules for communicating, planning, scheduling, and monitoring the execution of tasks and requests from other agents. We use the RETSINA structure, because it entails all types of functionality we think are required in TDA's.

Figure 3 outlines our TDA architecture. Alike the RETSINA architecture we differentiate between four different types of agents: interaction agents, task agents, information gathering agents and information management agents. Each segment in the figure consist of a set of agents that jointly carry out the overall task. The interface agents interact with the users, receive user input, and take care of display functions. The interface agents are closely tied (and tuned) to the goals of the user. The task agents perform the more complex tasks such as making problem-solving plans and carrying out these plans by coordinating and exchanging information with other software agents. The information agents provide intelligent access to a variety of information sources, and are, logically, closely tied to the available information sources. The task agents carry out most of the problem solving processes, and are therefore more complex in nature than the interface and information agents. They have knowledge of the task domain and have a good understanding of who their problem-solving counterparts are: which other agents are relevant to the problem solving task at hand. Worth noticing is that the task agents are discriminated by their role in the C2 model mentioned earlier. This is done to emphasize the large variety of tasks that are present in a C2 process, ranging from fairly simple skill based tasks (such as SA and DC tasks) to complex knowledge based tasks

(TA and DM). Finally, the information management agents perform all tasks related to managing information and services among the agent-community. They act as proverbial middlemen: they provide domain-independent services that allow the other agents to function properly. Figure 3 displays some of the tasks that belong to this category, such as information distribution, information integration and task coordination. The ontologies present in the architecture provide domain specific knowledge to the domain tasks are managed by information management agents. The use of ontologies will be detailed later on.

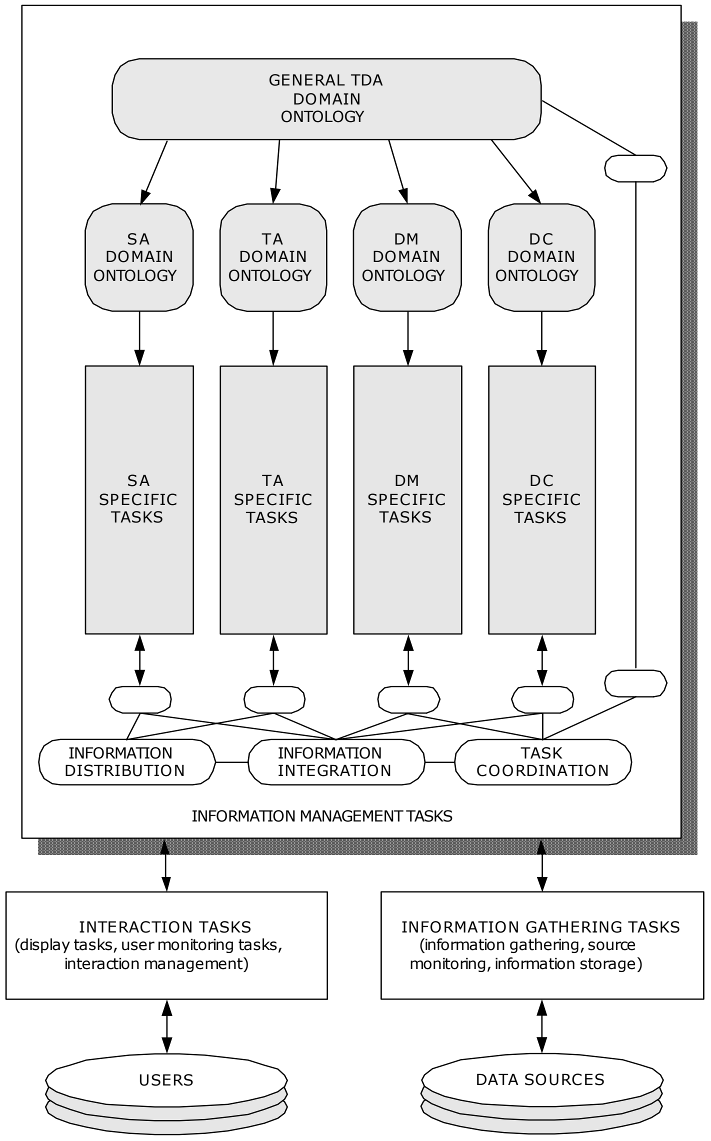
GENERAL TDA DOMAIN ONTOLOGY
SA DOMAIN ONTOLOGY
TA DOMAIN ONTOLOGY
DM DOMAIN ONTOLOGY
DC DOMAIN ONTOLOGY
SA SPECIFIC TASKS
TA SPECIFIC TASKS
DM SPECIFIC TASKS
DC SPECIFIC TASKS
INFORMATION DISTRIBUTION
INFORMATION INTEGRATION
TASK COORDINATION
INFORMATION MANAGEMENT TASKS
INTERACTION TASKS (display tasks, user monitoring tasks, interaction management)
INFORMATION GATHERING TASKS (information gathering, source monitoring, information storage)
USERS
DATA SOURCES

Figure 3: A Generic Architecture for TDA Design.

The reason for using such architecture stems from task analysis: the architecture adheres to the notion that one can distinguish between generic and domain-specific tasks. During an in-depth typical task analysis one may arrive at a level of detail at which further decomposing a tasks yields basic, domain independent tasks such as ‘information retrieval or ‘display message’, etc. Although the subject of these tasks is instance-specific (i.e. 'retrieve the state of sensor x from database y'), the task themselves are generic. In contrast, there are tasks that are specifically domain-dependent, such as tasks in which specialized algorithms are applied ('determine the coverage of radar x at moment y'). These tasks are characterized by the large volume of domain knowledge that they require. We can apply this notion in our agent-oriented design philosophy: we differentiate between agents that provide domain-independent *services* and agents that provide *domain-specific* functionality. The following table shows proposed types of agents along with their respective dependence on domain-knowledge.

category	provides	domain dependence	reusability
information gathering agents	components that provide access services to data sources (such as databases or networks)	low, tasks are generic across all types of systems	high, can largely be designed in a domain independent manner
interaction agents	components that provide user interaction services (such as display, user input management, user monitoring)	moderate, specific functionality needs to be adapted to fit cognitive user requirements	moderate to high, reusability lies in the use of generic building blocks
information management agents	components that distribute and preprocess information and deal with task management	moderate, basic functions are generic, but need to be tuned towards technical requirements	moderate to high, reusability lies in the use of generic building blocks
domain specific task agents	components that deal with domain specific tasks, such as specialized information analysis processing or employing certain reasoning strategies	high, tasks are only applicable to specific systems	low, although generic problem solving methods may be used

The above table illustrates some of the opportunities for reusability for each of the classes we use. Logically, domain dependence is inversely correlated to reusability. If a certain task requires a high amount of domain knowledge, chances are slim that the component that performs this task can be employed in another domain setting. Conversely, domain independent tasks are likely to be applicable in a wide range of support systems with minimal reconfiguration effort and therefore are highly reusable.

4.2 Reusability in TDA Design

The importance of emphasizing reusability becomes even obvious if we take a look at the actual task of building the various components of a TDA. One of the reasons we advocate the use of a framework such as RETSINA, is that each agent is made up of the same structure. The task of building an agent more or less becomes a case of *instantiation* of the appropriate blueprints instead of construction. As for the issue of domain-dependence, the focal point is on building the parts of each agent that directs task execution. For the service level agents, designing the behavior is very much straightforward. Tasks such as accessing information sources or capturing user-input can be accomplished by simple rule-based behavior and can be designed without too much user involvement. Typical tactical decision making tasks however usually reside at the level of knowledge-based tasks (Rasmussen, 1986), and therefore require intensive user-centered design procedures. The research field that specifically addresses this issue is the field of knowledge engineering. There are two interesting concepts from this field we can use: domain ontologies and generic problem solving methods (Schreiber et. al., 2000; Perez and Benjamin, 1999; Motta et. al., 1999). Domain ontologies specify domain specific knowledge at a generic level, which can be shared by multiple reasoning components communicating during a problem solving process. Problem Solving Methods (PSMs) describe in a domain-independent way the generic reasoning steps and knowledge types needed to perform a task. Both ontologies and PSMs offer promising opportunities for achieving reusability and can be used in complementary manner to derive new knowledge intensive systems from existing, reusable components. Since most C2-specifc tasks can be considered complex knowledge-based tasks, we believe ontologies- and PSMs may be well applicable in this case.

Complex tasks often involve stereotypical problem-solving behaviors. These behaviors (or methods) can be specified by means of patterns and subsequently provide standard ways of addressing certain kinds of application tasks. For instance: a medical diagnosis task uses a lot of the same reasoning patterns as technical diagnosis tasks. By explicitly modeling the generic reasoning strategies we obtain reusable blueprints for complex tasks. A task description states what is to be done, and the PSMs describe how it should be done. The PSMs specify which inference actions (reasoning primitives such as forward and backward chaining, matching, select and search) have to be carried out for solving a given task and also determine the sequence in which these actions have to be activated. In addition, so-called knowledge roles describe which role the domain knowledge plays in each inference action (Studer et al., 1998). Figure 4 illustrates the CommonKADS PSM for a monitoring task. PSMs are used to realize tasks by applying domain knowledge in a specific manner. The manner in which PSMs are applied and the role they play in the engineering process differs per methodology, but they all make use of a set of interconnecting models and descriptive languages, that jointly provide the basis for a knowledge based system. Several libraries of PSMs have been developed, most notably the CommonKADS library, which contains PSMs for diagnosis, prediction of behavior, assessment, design, planning, assignment and scheduling and engineering modeling (Schreiber et al., 2000) and the library detailed in (Benjamins, 1993), which is specifically aimed at diagnosis. Some other collections of PSM can be found in (Motta, 1998), (Marcus, 1988) and (Chandrasekaran, 1990).

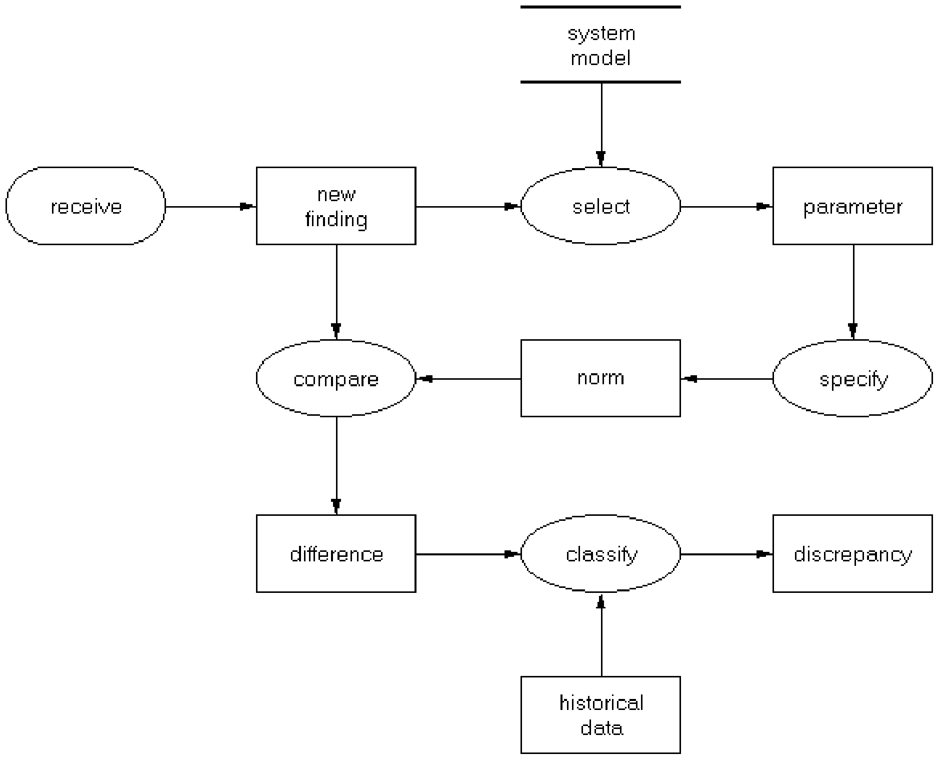

Figure 4: The CommonKADS PSM for a Monitoring Task (Schreiber et al, 1998).

The other concept, ontologies, defines the basic terms and relations of a topic area as well as the rules for combining terms and relations to define extensions to the vocabulary (Neches et al., 1991). Ontologies are meant to capture domain knowledge in such a generic way that it may be reused and shared across applications and systems. Ontologies are, in that sense, somewhat similar to taxonomies, but have a much richer internal structure and additionally, reflect *consensus* about the knowledge in that domain between all parties that make use of that ontology. Ontologies are generally used in the construction of domain models, which, in turn, are used in the construction of knowledge based systems. Ontologies come in different flavors. They all capture static domain knowledge in an application independent way, but can vary in their level of generality. For instance, *domain ontologies* are applicable within a certain domain (e.g. the naval C2 domain), while *generic ontologies* are applicable across a range of domains (e.g. in the land, naval and air C2 domain). Building an ontology for a certain domain requires an in-depth analysis of that domain, because all relevant concepts, attributes, relations, constraints, instances and axioms need to be uncovered (Studer et al., 1998). A more detailed classification of ontologies can be found in (Guarino, 1997), which is especially meant for information processing systems, such as decision support systems. Guarino distinguishes between four different kinds of ontologies. *Top-level ontologies* describe generic concepts independently of a particular domain or problem. A top-level ontology contains descriptions of abstract kinds of concepts, such as time, space, event, and action. *Domain ontologies* describe the terminology of a generic domain. This would include statements like ‘a submarine is a subtype of sea-platform’ or a description of the capabilities of a certain frigate type. *Task ontologies* describe specific terminology for generic tasks or activities. For instance, if we are concerned with diagnosis, we will talk about ‘hypotheses’, ‘symptoms’ and ‘observations’. We say that those terms belong to the task ontology of the diagnosis task. An *application ontology* describes concepts from both the domain ontology and the task ontology and describes the roles domain entities play within the system.

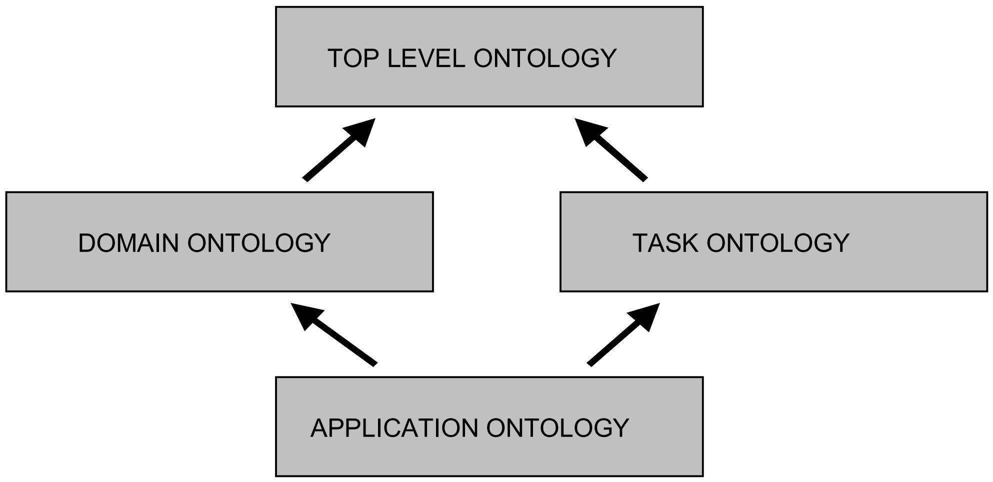

Figure 5: Types of Domain Ontologies (Guarino, 1997). The Arrows Denote Subtypes (i.e. A domain ontology is a subtype of the top-level ontology).

Ontologies are usually an integral part of knowledge engineering methodologies. Together with various models that describe various views on the domain and system, they form the blueprint for a system. Ontologies are widely used and applications of ontologies can be found in lots of different system domains, such as knowledge engineering and management, natural language generation systems, knowledge-based systems, and multi-agent systems. Also, a widely applied use for ontologies is to use them to achieve interoperability between systems.

We believe these two concepts, problem solving methods and domain ontologies, are key methods in designing agents that perform the complex problem solving tasks that are present in tactical decision making. Even though knowledge engineering is primarily focussed on the development of expert systems, there have been several efforts to incorporate the concepts from this field in the design of distributed problem solving architectures such as ComoMAS (Glaser, 1997), DESIRE (Brazier et al., 1997) and MAS-CommonKADS (Iglesias et al., 1998). We will outline the basic development process for distributed agent-oriented TDA design in the following section.

4.3 The Overall TDA Development Process

We have addressed several aspects of our proposed methodology. We have touched upon design requirements (reusability, user-centered design), architectural concepts (using an agent-oriented design philosophy) and, knowledge engineering concepts (problem solving methods, domain ontologies), but we have not detailed how we think these themes can be used in an integrated design methodology. This section outlines the basic TDA development process using the ideas and concepts from the previous sections.

The COADE methodology especially addresses the interface agents and the agents in the information management layer, i.e. *how* the support should be presented, how the user should be able to interact with the system and the necessary information management tasks that enable these requirements. On a more conceptual level, the COADE strategies lay out the fundaments for the agent organization.

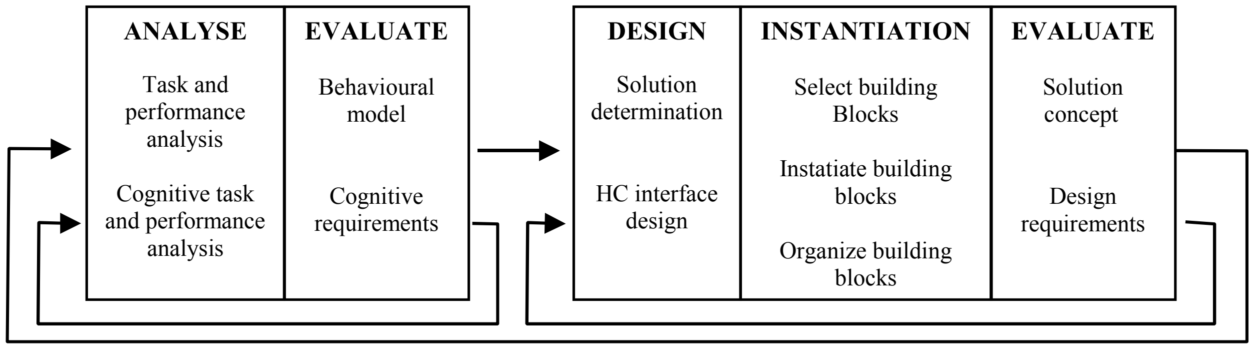

Figure 6: COADE Framework with Added Agent Organisation Phase.

The design process consists of three basic phases, selecting, instantiating and organizing the building blocks that make up the TDA.

Phase 1: Selecting Appropriate Building Blocks

Given the general behavioral and cognitive requirements as determined by the various COADE processes, we can begin with selecting appropriate and applicable agent templates. Since we already have an idea of how the support system should look like (stemming from the DESIGN phase of COADE), the task of building a rudimentary agent organization is a relatively easy task. We know which information sources should be available, which information accessibility requirements are present and which decision-making tasks should be performed by the support system. To this end, we devise a crude layout of the agent organization and subsequently select the blueprints for information agents, interface agents and task agents.

Phase 2: Instantiating Building Blocks

The blueprints we have selected must be tuned towards the TDA we are designing. For the service-level task agents (e.g. the information gathering agents and interface agents) instantiation is comprised of mainly setting the right parameters (e.g. designating the information source or setting display formats), due to the generic nature of these task agents. For the task agents (the agents that perform the domain-specific decision making tasks) we employ the basic building templates for task agents, but instantiate them by selecting appropriate problem solving methods and domain ontologies. This process proceeds in the same way knowledge engineering methodologies such as CommonKADS do, but uses task information gathered from the COADE methodology.

Phase 3: Organizing Building Blocks

Since we now have a general picture of the agents involves and the internal layout of the agents, we now have to turn to the organization of these agents, i.e. how they should cooperate and how information and messages should be passed around. This involves in instantiating the information management agents and appropriately configuring the communication parts of each agent, so that they jointly form the TDA.

The use of agent frameworks and smart use of reusability centered concepts from knowledge engineering thus yield interesting opportunities for fast development of prototypes. Of course, the route to a ready to deploy TDA is a long one, but there are plenty of possibilities to make the development process somewhat easier.

5.0 A CASE STUDY: SENSOR MANAGEMENT

We have applied the basic steps of our concept on a small case study: designing a tactical decision aid for sensor management. Sensor management is a very broad problem domain and is comprised of numerous tasks related to the proper allocation and scheduling of sensors to environmental requirements. The aim of this case study was not to actually design a TDA, but to get a feeling for the applicability of such a generic design approach to a actual tactical decision making problem. Therefore, the case study is neither complete, nor very detailed. The following sections describe the various steps that one would perform using the methodology. In an actual TDA development process, one would iterate the process a number of times to obtain a more detailed TDA model every cycle. Due to time limitations in this case study the process was only conducted for a single iteration (i.e. the process stopped after the Evaluation phase) and does not contain any research on the agent design process.

5.1 Step 1: Task Analysis

Earlier in the paper a two-level model of command and control was introduced. Sensor management is an important part of the function *Situation Awareness*. Many of the generic commands, such as air surveillance and the identification command, contribute to the building of the Common Operational Picture. A number of commands are related to the function Direction and Control, for example missile and gunfire support commands.

Sensor management can be decomposed into four different subtasks:

1) Monitor sensors: the generation of commands for sensor diagnosis, sensor coverage testing, or re-allocation of tasks allocated to malfunctioning sensors. Sensors are monitored continuously to test their performance and detect possible degradation. When a sensor is malfunctioning, a number of commands can be initiated for testing purposes. Examples are sensor diagnosis and sensor coverage

commands. Another subtask of ‘monitor sensors’ is starting the reallocation of commands that are currently allocated to sensors that are not functioning properly.

2) Specify commands: the generation of commands that have to be performed given the directives or the tactical situation. During operation, commands are generated for tactical reasons. Some of the commands are initiated because of duties and directives determined by the task group’s mission or by picture compilation orders. Other commands are initiated because of changes in the tactical situation or are specified by ZIPPO’s. Fifteen generic sensor commands are distinguished in this study. Each sensor command has a priority, a start time, a duration and a type (passive or active).

3) Allocate commands: the commands are ordered by priority. Next, for each command all sensors are listed that could perform the command. Taken into account are EMCON restrictions and availability of the sensors. When commands have to be allocated to sensors, the commands with the highest priority are allocated first. The reason for using priorities is that, due to a limited capacity, sometimes not all commands can be allocated immediately and commands with a higher operational priority take precedence. In the subtask ‘pre-allocation’, the generic commands are mapped to the sensors that are able to perform them. EMCON restrictions and technical availability are considered in this allocation process. Additionally, possible dependencies or interference between the commands have to be detected.

4) Plan sensors: the tasks are mapped onto a sensor, based on capability of the sensors and performance computation. When the allocation process is finished, the generic commands are translated to sensor-related versions of these commands. For example, a generic ‘air surveillance command’ can, amongst others, be matched to APAR_horizontal_search, APAR_limited_volume_search, SMART-L_air_operational_mode, SIRIUS_air_search_narrow, and so on. Each of these sensor-related commands has a number of attributes such as power, range, pattern or angle. In the end, suitable sensor-related commands are produced. For each sensor-related command the expected performance of the sensors is predicted based on environmental and geographical data. For active sensors, performance is expressed as a combination of detection range and counter-detection range. Passive sensor performance is expressed in detection range only. The command is assigned to the sensor with the highest predicted performance. A command can be assigned to another sensor if too many commands are assigned to one particular sensor or if conflicts between sensors arise. Not all commands require performance computing. Finally, the final command assignment is made based on the already assigned commands, the possible allocations of the commands and the performance estimates.

Task Characteristics and Performance

Some of the generic sensor commands are directly related to the safety of the ship and, hence, have a high priority. These commands should be executed first, commands of lesser importance can be postponed. Quick recognition and selection of the urgent commands is essential in sensor management. In the succeeding development cycles, experts should be asked to determine time and quality constraints.

Person Characteristics and Performance

The user of the TDA must be a trained expert on sensor management. Although parts of the process are likely to be automated, understanding of the sensor management process is required. In order to come to a sensible and usable TDA for sensor management, the person characteristics (e.g. required education, level of expertise, style and attitude) need to be uncovered or established.

Environmental Characteristics and Performance

Sensor management tasks are primarily single-user tasks, even though several people may be involved in overall process. Team organization, therefore, is fairly simple. In the following development cycles, these and other conditions under which the sensor management task is performed need to be examined in detail.

5.2 Step 2: Decision-Making Characteristics

Because the TDA is designed using a description of the sensor management process instead of studying the situation in reality, unfortunately no analysis of characteristics of operator decision-making was possible at this stage of the development process.

5.3 Step 3: Cognitive Task Analysis

Most subtasks in sensor management are of medium complexity. In Rasmussen's SRK-framework (Rasmussen, 1986), they would be classified as rule-based. The pre-allocation and the 'assess commands' subtasks appear to be the most complex and time-consuming tasks. Pre-allocation requires combining EMCON restrictions, the radar frequency plan, sensor availability, and possible dependencies between allocations. Also for the command assessment subtask several types of information have to be merged by the operator: the sensor related commands and their performance, commands already allocated to the sensors and sensor capability restrictions. Especially in high-demand situations these subtasks may become problematic.

5.4 Step 4: Solution Concept

The complexity of the sensor suite on board of the ADCF is high. The APAR and SMART-L can be used in many different ways and for many different tasks. It is suggested that because of the complexity, the operator responsible for sensor management should be aided by software.

Five different ways of support are distinguished in the TDA framework. In the 'manual-mode' the user has complete authority and the system remains complete passive. A more active system would give the user advice, or even work co-operatively with the user. In 'automatic mode' the system has total authority, the operator only has the power to overrule decisions. The final stage is a complete independent system without any user-interface, a so-called 'black box'. For each of the subtasks defined as being tasks that need to be supported, the appropriate mode of support must be chosen:

- Initiation of sensor monitoring commands is most likely a task of the user, depending on the quality of the fault diagnosis process of the systems.
- Initiation of the larger part of the generic sensor commands is also the work of the operator. Some commands however are generated automatically by the CMS. Especially quick reaction commands (ZIPPO's) are pre-fabricated.
- The commands that have to be allocated to the sensors should be presented to the operator ordered by priority. This diminishes the chances of overseeing urgent commands.
- Pre-allocation can be solved in a rule-based manner, but there is a chance that opportunities are missed. Complex systems like APAR or SMART-L may sometimes be degraded and hence only partly available with respect to functionality. What can and cannot be done when a system is degraded depends strongly on the functionality that is missing. Deciding which commands can be allocated to degraded systems is, for the time being at least, a task for the operator. When the operator is overloaded, the task can be automated in a sub-optimal way by interpreting degraded as not available.

- The specification of sensor-related commands is a rule-based task. The rules are described in the 'ADCF model of expertise sensor control' manuscript. This step can be automated.
- Prediction of sensor performance is complex. Because both detection and counter-detection are part of the performance computation, it is a multi-criteria optimization. In the near future this can be automated partly by existing software, but expert knowledge remains required to include all the nuances. Allocation of this task could depend on the workload of the operator.
- Final assessment of the commands is a complex subtask. Interference between new commands and already assigned commands has to be detected and multi-criteria performance measures have to be combined. User-involvement seems required for this subtask.

5.5 Step 5: User Interface Design

In the first pass through the TDA framework cycle, interface design only needs to be considered at a low level of detail. Only HCI design on the task level is required. Communication level design is only appropriate when a certain level of detail has been described and the first hand-on experiments are required.

5.6 Expert Evaluation of the Case Study

Feedback from experts is required to determine whether these first concepts match the needs and desires of future users. For this purpose, two experts where asked for comments. One expert is a leading scientist on performance of radar systems, the other expert works for the Dutch Royal Navy.

Both experts were content with the results of the first ANALYSE-DESIGN-phase, and provided some suggestions to improve the concept. Part of the task "specify sensor-related commands" is to determine correct and suitable sensor parameter settings. Modern sensor systems, and especially APAR, have a huge number of parameters that must be set for optimal performance. This is an extremely complex task that requires deep knowledge of the system. Due to reduction in staffing size, most operators will be generalists who will lack knowledge about important details. Therefore, this task, a little overlooked in the current concept, should be supported by the TDA.

According to the two experts, high workload in not a great issue in sensor management. There is hardly any time pressure, and task-set changes are therefore not a problem either. On the other hand, complexity is. Degradation of sensors, interference and dependencies between commands, performance computation and determining optimal settings for sensor systems on the LCF are all complex tasks. Sensor management support should focus on these issues.

Visualization is an important tool to help the operators cope with complexity. Performance and detection ranges could be visualized in a way that depends on the task and context. The current system, Pirate, does not correspond well with user needs.

Another aiding principle suited for the Sensor Management TDA is knowledge-based support on the effects of interference between commands, and on temporal consequences of applying a sensor to a specific task. Also, a knowledge-based tool could model the remaining capabilities of degraded systems.

6.0 CONCLUSIONS

The design methodology presented in this paper combines a user-oriented design with a generic technical architecture. Even though there is still a lot of ground to cover in terms of detailing and formalizing the approach, the integral approach seems very much worthwhile. The introduction of COADE has shown to be very valuable, because it sets forth essential functional requirements for good TDA design. The development of a true generic technical framework seems feasible, in part due to the use of an agent-oriented architecture, which provides the necessary flexibility and task-oriented style we require.

A lot of research still needs to be done though. The methodology needs to be investigated and refined. We need to describe the steps in more detail and define the specifics that set it apart from COADE. Also, we need to be able to specify and model the tactical aspects in more detail. Finally, the technical architecture needs to be developed. Until now, the architectural framework is still conceptual, but to prove its worth, we need to look into actually providing agent-blueprints that can be used in case studies. Nevertheless, as an integrated approach, the overall methodology seems worthwhile and offers a promising and innovative approach to user-centered TDA design.

7.0 REFERENCES

Benjamins, V.R. (1993). *Problem Solving Methods for Diagnosis*. PhD thesis, University of Amsterdam, Amsterdam, The Netherlands.

Brazier, F.M.T., Dunin-Keplicz, B.M., Jennings, N.R. and Treur, J. (1997). *DESIRE: Modelling Multi-Agent Systems in a Compositional Formal Framework*. International Journal of Cooperative Information Systems, 6(1), pp. 67-94.

Chandrasekaran, B. (1986). *Generic Tasks in Knowledge Based Reasoning: High Level Building Blocks for Expert System Design*. IEEE Expert, 1(3), pp. 23-30.

Chandrasekaran, B. (1990). *Design Problem Solving: A Task Analysis*. AI Magazine, 11, pp. 59-71.

Chalmers, B.A. & Burns, C.M. (1999) *A Model-Based Approach to Decision Support for a Modern Frigate*. Proceedings of the TTCP Symposium on "Coordinated Battlespace Management", Space and Naval Warfare Systems Center, San Diego, CA, USA, 1999.

Decker, K. and Sycara, K. (1997). *Intelligent Adaptive Information Agents*. Journal of Information Systems, 9, pp. 239-260.

Endsley, M.R. & Kiris, E.O. (1995). *The Out-of-the-Loop Performance Problem and Level of Control in Automation.* Human Factors, 37 (2), pp. 381-394.

Essens, P.J.M.D., Fallesen, J.J., McCann, C.A., Cannon-Bowers, J. & Dörfel, G. (1994). *COADE, A Framework for Cognitive Analysis, Design and Evaluation,* NATO Defence Research Group, Panel 8 on Defence Applications of Human and Bio-Medical Sciences/RSG.19 on Decision Aids in Command and Control, July 1994.

Glaser, N. (1997). *The CoMoMAS Methodology and Environment for Multi-Agent System Development,* In: C. Zhang, D. Lukose (eds.), Multi-Agent Systems – Methodologies and Applications, LNAI 1286, Springer-Verlag, Berlin, Germany, 1997, pp. 1-16.

Guarino, N. (1998). *Formal Ontology and Information Systems* In N. Guarino, editor, Proceedings of the 1st International Conference on Formal Ontologies in Information Systems, FOIS'98, Trento, Italy, IOS Press, pp. 3-15.

Iglesias, C.A., Garijo, M., Gonzalez, J.C., Velasco, J.R. (1998). Analysis and Design of Multi-Agent Systems using MAS-CommonKADS. In Intelligent Agents IV (LNAI Volume 1365) Singh, M.P., Rao, A., Wooldridge, M.J., (eds.), Springer-Verlag: Berlin, Germany, pp. 313-326.

Kersholt, J.H. & Passenier, P.O. (2000). *Fault Management in Supervisory Control: The Effect of False Alarms and Support.* Ergonomics, 43 (9), pp. 1371-1389.

Klein, G.A. (1989). *Recognition-Primed Decisions*. In W. Rouse (ed), Advances in Man-Machine Systems Research, Greenwich, JAI Press. pp. 47-92.

Klein, G.A. (1993). *Naturalistic Decision Making: Implications for Design.* SOAR 93-1: Crew Systems Ergonomics Information Analysis Centre, Wright-Patterson AFB, USAF, OH.

Marcus, S. (1988). *Automatic Knowledge Acquisition for Expert Systems*. Kluwer Academic Publishers.

Motta, E., Fensel, D., Gaspari, M., Benjamins, R. (1999). *Specifications of Knowledge Components for Reuse*, Proceedings of 11th International Conference on Software Engineering and Knowledge Engineering, Kaiserslautern, Germany, 1999, KSI Press, pp. 36-43.

Motta, E. (1999). *Reusable Components for Knowledge Models*. IOS Press, Amsterdam.

Neches, R., Fiches, R.E., Finin, T., Gruber, T.R., Patil, R., Senator, T. and Swartout, W. (1991). *Enabling Technology for Knowledge Sharing*. AI Magazine, 12, pp. 36-56.

Neerincx, M.A., van Doorne, H. and Ruijsendaal, M. (1999). *Attuning Computer-Supported Work to Human Knowledge and Processing Capacities in Ship Control Centers.* In J.M. Schraagen, S.F. Chipman, V.L. Shalin (Eds.) Cognitive Task Analysis, Lawrence Erlbaum Associates, Inc.

Neerincx, M.A., Ruijsendaal, M. and Wolff, M. (2001). *Usability Engineering Guide for Integrated Operation Support in Space Station Payloads,* International Journal of Cognitive Ergonomics, 2001, 5(3), pp. 187-198.

Passenier, P.O. & Delft, J.H. van (1995). In Delft, J.H. van & Schuffel, H. (Eds.) *Human Factors onderzoek voor toekomstige Commando Centrales KM (Rapport TNO-TM 1995 A-19).* Soesterberg: TNO Technische Menskunde.

Pennington, N. and Hastie, R. (1993). *The Story Model for Juror Decision Making*. In R. Hastie (Ed.) Inside the Juror: The Psychology of Juror Decision Making. Cambridge: Cambridge University Press.

Perez, A.G. and Benjamins, V.R. (1999). *Overview of Knowledge Sharing and Reuse Components: Ontologies and Problem-Solving Methods*. IJCAI99 Workshop on Ontologies and Problem-Solving Methods (KRR5), Stockholm, Sweden, 1999.

Rasmussen, J. (1986). *Information Processing and Human-Machine Interaction: An Approach to Cognitive Engineering*. North Holland Series in Systems Science and Engineering, Vol. 12, New York, North Holland.

Schreiber, A.T., Akkermans, J.M., Anjewierden, A.A., De Hoog, R., Shadbolt, N.R., Van de Velde, W., and Wielinga, B.J. (2000). *Knowledge Engineering and Management: The CommonKADS Methodology*. The MIT Press, Cambridge, MA.

Studer, R., Benjamins, V., and Fensel, D. (1998). *Knowledge Engineering: Principles and Methods*. IEEE Transactions on Data and Knowledge Engineering, 25, pp. 161-197.

Sycara, K. and Zeng, D. (1996). *Coordination of Multiple Intelligent Software Agents.* International Journal Cooperative Information Systems, 5(2/3), pp. 181-212.

Willeges, R.C., Williges, B.H. and Elkerton, J. (1987). *Software Interface Design.* In Salvendy, G. (Ed.) Handbook of Human Factors. New York: John Wiley & Sons.

Woods, D.D. and Roth E.M. (1988). *Cognitive Systems Engineering.* In Helander, M. (ed.), Handbook of Human-Computer Interaction, pp. 3-43. Elsevier Science Publishers B.V., North-Holland.