DOCUMENT CONTROL SHEET

	ORIGINATOR'S REF.		SECURITY CLASS.
	TP 96021 U		Unclassified

ORIGINATOR
National Aerospace Laboratory NLR, Amsterdam, The Netherlands

TITLE
Application of distributed artificial intelligence in complex modular critical applications

PRESENTED AT

AUTHORS	DATE	pp	ref
R.G. Zuidgeest	950915	61	102

DESCRIPTORS

Artificial intelligence	Man machine systems	Task complexity
Decision making	Man-computer interface	
Distributed processing	Mission planning	
Expert systems	Modularity	
Flight control	Real time operation	
Funtional analysis	Support systems	

ABSTRACT
This report provides an overview of the emerging technology Distributed Artificial Intelligence, in particular in the area of Distributed Problem Solving (DPS). DPS refers to coarse-grained (task-level) problem decomposition resulting in a number of expert or knowledge-based systems, generally called agents of which each exhibits some intelligence.
The DPS technology has features that may reduce system design complexity through a highly modular approach and, consequently, may reduce life cycle costs through improved maintainability. These problems of complexity and maintenance are often faced with the design of complex critical applications (including many aerospace applications). DPS can provide a more natural solution with respect to system design, development, and maintenance.
This report surveys DPS methods and techniques that have potential benefit for these critical applications. The two main approaches in DPS are discussed: blackboard systems and multi-agent systems. Further, the technology is evaluated along a number of criteria relevant for the envisaged applications. Based on this evaluation it is recommended to consider DPS technology in complex modular (decomposable) critical systems and let it be a driving technology for the overall system architecture.

NLR TECHNICAL PUBLICATION

TP 96021 U

APPLICATION OF DISTRIBUTED ARTIFICIAL INTELLIGENCE IN COMPLEX MODULAR CRITICAL APPLICATIONS

by

R.G. Zuidgeest

This report is based on research performed by NLR in the context of the EUCLID CEPA-6 RTP 6.5 "Crew Assistant" Project. This research has been carried out partly under contract awarded by the Royal Netherlands Air Force representing the members of the RTP 6.5 Management Group (the contract number is DMKLu 135/94/9419/11) and partly through NLR's own research funding.

Division : Informatics
Prepared : RGZ/ 7/2-96
Approved : FJH/ 8/2

Completed : 950915
Order number : 555.405/555.407/107.408
Typ. : JvE

Summary

This report provides an overview of the emerging technology Distributed Artificial Intelligence, in particular in the area of Distributed Problem Solving (DPS). DPS refers to coarse-grained (task-level) problem decomposition resulting in a number of expert or knowledge-based systems, generally called agents of which each exhibits some intelligence.

The DPS technology has features that may reduce system design complexity through a highly modular approach and, consequently, may reduce life cycle costs through improved maintainability. These problems of complexity and maintenance are often faced with the design of complex critical applications (including many aerospace applications). DPS can provide a more natural solution with respect to system design, development, and maintenance.

This report surveys DPS methods and techniques that have potential benefit for these critical applications. The two main approaches in DPS are discussed: blackboard systems and multi-agent systems. Further, the technology is evaluated along a number of criteria relevant for the envisaged applications. Based on this evaluation it is recommended to consider DPS technology in complex modular (decomposable) critical systems and let it be a driving technology for the overall system architecture.

Contents

(61 pages in total)

Abbreviations

AI	Artificial Intelligence
AIP	Advanced Information Processing
DAI	Distributed Artificial Intelligence
DPS	Distributed Problem Solving
KBS	Knowledge-Based System
OO	Object-Oriented

1 Introduction

This report contains the result of a study on distributed artificial intelligence (DAI) as a sub-area of advanced information processing (AIP) technologies from which complex modular critical applications can benefit. Many advanced applications in aerospace, military and civil domain fall under this category. In order to precise the relevant category of applications, the adjectives "complex", "modular", and "critical" need explanation.

A "complex" application relates to one or more of the following characteristics:
- Large (wrt hardware and software as well as operational environment).
- Heterogeneous (incorporates different components).
- Complex interactions (internal as well as external).
- User interaction at a high cognitive level (abstract information and control, decision support).
- Degree of autonomy (exhibits autonomous behaviour).
- Intelligence (exhibits intelligent behaviour like an expert system).

A "modular" application consists of well-identifiable components or sub-systems according to some dimension. These components are often called "agents" in the context of DAI technology or "module" in general. In this context, modularity expresses the distributed or decomposable character of an application. Typical dimensions of distribution are geography (system components are located at different places), functions (the system consists of different functional components), etc.

A "critical" application relates to safety, survivability, real-time, and interaction with or operation in a dynamic, potentially hazardous environment.

DAI technology has its benefits to a broad spectrum of these type of applications. To mention a few: robotics, command and control, multi-sensor data fusion, on-board crew assistant, computer networks, and planning systems. Whenever "application" or "system" is used in the text, it is meant to be this type of applications.

The report focuses on a subfield of DAI: distributed problem solving DPS (including distributed expert systems), as opposed to parallel artificial intelligence, which concerns connectionism (e.g. neural networks). The DPS technology has features that may reduce system design complexity through a highly modular approach and, consequently, may reduce life cycle costs through improved maintainability. Life cycle problems are often faced with complex critical applications and DPS can provide a more natural solution. DPS methods and techniques that have potential

benefit for these applications are surveyed on basis of recent literature reflecting state-of-the-art DPS.

Section 2 provides context of the DPS technology, which includes the presentation of a taxonomy of fields, the motivation of the importance of DPS, and the discussion of a number of typical aspects or dimensions of DPS applications. Section 3 discusses in more detail the functionality offered by DPS technology in terms of decomposition, distribution, cooperation and architectures. These starting sections should provide a solid base for a detailed assessment of DPS technology with respect to a list of criteria as is done in section 4. Finally, section 5 provides concluding remarks on the offered functionality.

2 Context

This section provides an introduction to the technology area Distributed Artificial Intelligence, and in particular in the area of Distributed Problem Solving (DPS). The technology is explained and a motivation of employing this technology is provided. Section 2.1 discusses a taxonomy of DAI technology and identifies the place of DPS by means of a taxonomy and hence defines the scope of this report. Section 2.2 presents a number of potential benefits for the relevant applications. Section 2.3 provides a schema to classify an application along a number of dimensions typical for the technology.

2.1 Taxonomy and scope

Distributed Artificial Intelligence (DAI) addresses distributed problem solving by multiple cooperative processing elements. It is concerned with issues of coordination among concurrent processes at the problem-solving and representation levels.

The following definition of DAI is adopted from [Dec87]:

DAI is concerned with solving problems by applying both AI techniques and multiple problem solvers.

DAI differs from the more general area of distributed processing, because it is concerned with distributing control as well as data and can involve extensive cooperation between entities [Mar92]. Distributed processing systems address the problem of coordinating a network of computing agents to carry out a set of separate and mostly *independent tasks*, as opposed to DAI. Distributed processing focuses on how bits of data can be physically moved among machines. So distributed processing or programming such as client-server are out of the scope of DAI and this report.

In order to provide a good scope of this report, Figure 1 shows a taxonomy of DAI [Dec87, Bon88b]. Two categories of DAI research exist: parallel artificial intelligence and distributed problem solving (DPS). Parallel AI refers to a fine-grained efficiency-oriented approach, also referred to as connectionism. Neural networks are an example of it. DPS refers to coarse-grained (task-level) problem decomposition resulting in a number of expert or knowledge-based systems, generally called agents. Each of these entities include or exhibit some intelligence, whereas parallel AI systems consist of entities that are relatively simple in construction and do not exhibit any intelligence, but the overall system exhibits some intelligence based on patterns of data processing of these fine entities (e.g. neurons in neural networks).

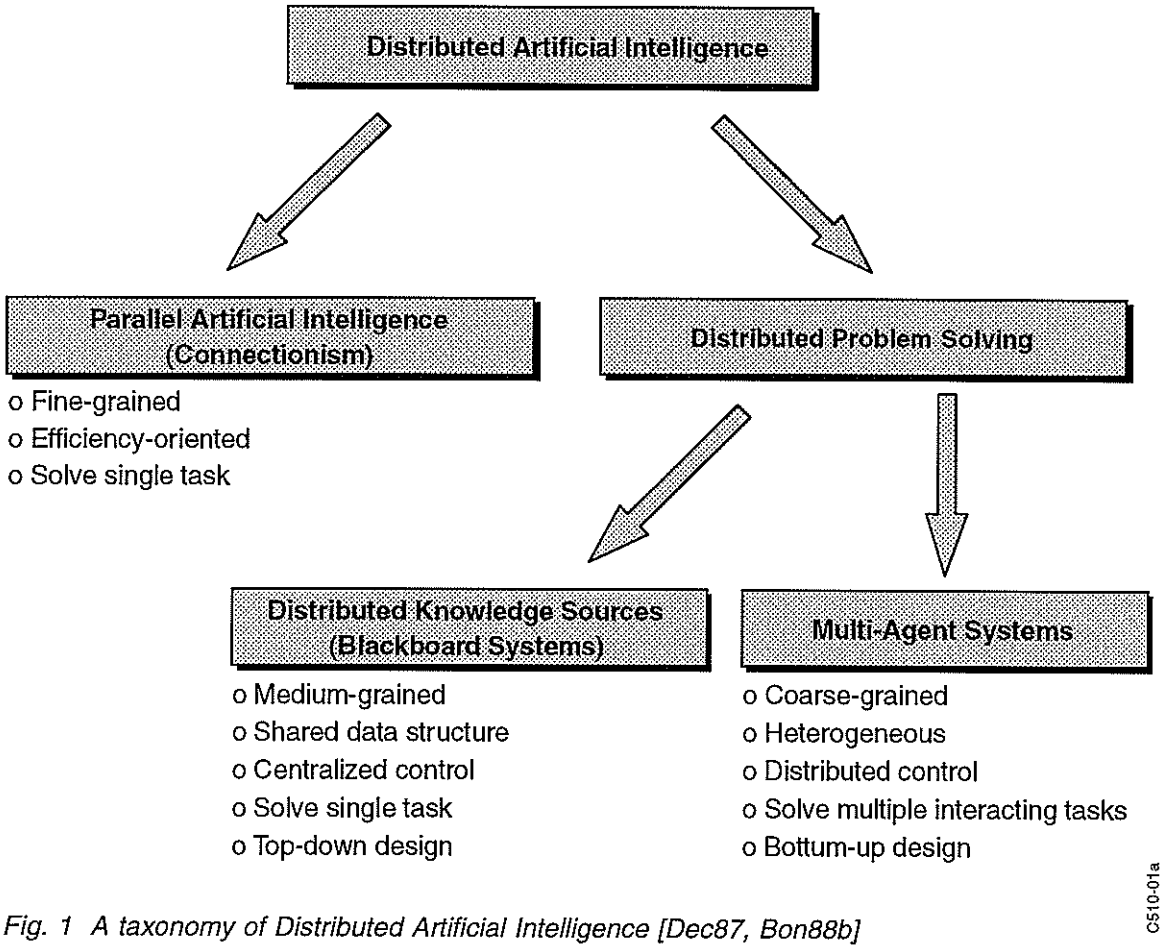


Fig. 1 A taxonomy of Distributed Artificial Intelligence [Dec87, Bon88b]

This report focuses on DPS. As shown in the figure, DPS can be divided into distributed knowledge sources (often referred as the blackboard system) and multi-agent systems. The latter normally consists of agents that have a range of expertise (e.g. complete knowledge-based systems) or functionality that have the potency to function stand-alone, as opposed to medium-grained knowledge sources in a blackboard system. The figure depicts a number of typical characteristics of both approaches. Section 3.4 will discuss these in detail.

In spite of these different properties, multi-agent and blackboard system technology have much aspects in common and therefore will be surveyed under the denominator DPS. Note that these do not exclude one another and can be both applied in systems whenever needed.

2.2 Motivation

Literature provides a rich set of potential advantages, reasons or merits that suggest, allow or stimulate application of DPS technology as listed below [Dur89, Mar92, Dec87, Bon88b].

Technological basis. The first reason is the technology push that provides DPS the technological basis. Hardware technology for processor construction and interprocessor communication has

become widely available and applied. Networks of relatively cheap processors and the technology of distributed processing [Sch90] are the basis for DPS application. These processor networks, either tightly or loosely coupled, provide the services [Tan89] for DPS applications.

The remaining arguments for employing DPS technology emerge from a market pull, where applications and application domains have inherent features or imposed system requirements that makes DPS technology a good candidate for implementation.

Inherent distribution. Many applications are inherently distributed. These applications and associated problems and tasks are better described as collections of separate agents (naturalness). Distributed applications may be:

- Spatially distributed (e.g. interpretation and integration of data from spatially distributed sensors).
- Functionally distributed (e.g. specialized agents on medical-diagnosis solving a difficult problem).
- Temporally distributed (e.g. production line in a factory).

Design and implementation benefits (modularity). The ability to structure a complex problem or task[1)] into relatively self-contained processing modules (agents) leads to a modular system. Each agent may be specialized in solving a particular aspect of the problem. Through cooperation among these specialized agents, a solution for the overall problem is found. Modularity allows a system to be constructed in a parallel, incremental and evolutionary way (both in the engineering phase and maintenance phase). It allows for scalability, extensibility, maintainability, and adaptability due to reduced complexity of agents performing a relative simple sub-task, and because knowledge and related processing is localized in a single expert or agent. This localisation enforces that only one agent (or a limited set) will be affected if certain domain knowledge or processing algorithms have to be revised, whereas the rest of the system will be left untouched.

Synergy (new classes of problems). There are problems which are too large for a centralized system, but can only be solved by cooperation of several independent (expert) systems. Truly intelligent systems contain so much knowledge that they must be broken down into multiple cooperating systems in order to be feasible.

Parallelism. If each agent in a distributed architecture is assigned to solve a specific aspect of the problem, it can be developed by specialists in that specific knowledge domain in parallel

1) The terms "problem" and "task" are used interchangeably. In the text, "solving a problem" may also be read as "performing/supporting a task".

with the other agents. Such an architecture allows also for parallelism and concurrency, the exchange of abstract information rather than raw data (henceforth reducing communication costs) and the placement of agents near sensing devices and devices to be controlled. These considerations potentially enhance real-time performance.

Integration. DPS technology allows for integration of existing heterogeneous computer systems that need to cooperate.

Reliability. DPS systems may be more reliable through redundancy, cross-checking, and triangulation of results. In this way, noisy, unreliable and uncertain data can be dealt with. With respect to system failure, where a centralized system fails completely, a DPS system may show graceful degradation if some agent or processor fails (adaption to failure) due to redundancy in communication paths and agents and modularity of design. [Les81] calls DPS systems dealing with these aspects "functionally accurate/cooperative systems".

2.3 Categorization and dimensions of DPS applications

This section provides a number of application areas for DPS and discusses a number of dimensions along which DPS applications can be characterized.

2.3.1 DPS application areas

[Dur89] categorizes (potential) DPS systems in four application areas:

- *Distributed Interpretation.* Distributed interpretation applications require the integration and analysis of distributed data to generate a (potentially distributed) semantic model of the data (e.g. multi-sensor data fusion [Zui94]).
- *Distributed Planning and Control.* Distributed planning and control applications involve developing and coordinating the actions of a number of distributed sensing and acting agents to perform some desired task (e.g. cooperating robots, distributed air-traffic control, command and control applications). Usually, data are inherently distributed among agents, having their own local planning database, capabilities and view of the world state.
- *Cooperating Expert Systems.* This application area deals with scaling expert systems technology to more complex and encompassing problem domains by developing cooperative interaction mechanisms to allow multiple expert systems to work together to solve a common problem.
- *Computer-Supported Cooperative Work.* Computer systems might overwhelm users with large amounts of information. By building AI systems that have coordination knowledge and consider users as special agents, users can be assisted in filtering the information and focusing attention on relevant information (e.g. intelligent command and control systems).

Note that many applications will be a mix of these areas.

2.3.2 Dimensions of DPS applications

[Sri87] defines a number of dimensions in which a work (e.g. study, application) can be classified. Table 1 provides these dimensions. It provides a good insight in the character of the application as a potential DPS application and its complexity.

Table 1 Classification of Crew Assistant as a DPS system

Dimension	Spectrum of values
System Model	Individual . . . . . .Committee. . . . . Society
Granularity	Fine . . . . . Medium . . . . . Coarse
System Scale	Small . . . . . Medium. . . . . Large
Agent Dynamism	Fixed . . . . . Programmable . .Teachable. . . . Autodidactic
Agent Autonomy	Controlled. . . . . Interdependent . . . . . Independent
Agent Resources	Restricted . . . . . Ample
Agent Interactions	Simple . . . . . Complex
Result Formation	By Synthesis . . . . . By Decomposition

System Model. Is the system a synthesis of a single intelligent agent from distributed (simple) components or an organization of multiple intelligent agents?
The robot application provides a good example on this question. A robot is a single intelligent agent that consists of a number of "distributed" or modular components: sensory components, an action planning component, various acting components, etc. These components can be considered as local agents themselves, consisting of hardware and software. At another level, an application may consist of multiple robots cooperating with each other while working on some objective. This application can be categorized as an *organization* of multiple agents.

Granularity. To what extent can the problem (data, task, communication packets, etc.) be decomposed?
Fine-granular systems consist of small but often many processing elements, whereas coarse-grained applications consist of large, complex, often intelligent and autonomous agents. An example of a fine-grained system is an image processing system where each agent processes a small, possibly overlapping part of the image (i.e. group of pixels) and communicates with

neighbour agents in order to process and interpret the global image. The robot organization discussed above is an example of a coarse-grained system.

System Scale. How many computing elements are employed, from a serial processor or a few (2-16) processor up to a million of elements on a connection machine?
Often for reasons of performance, granularity and system scale are antagonistic. The finer the granularity, the larger the system scale.

Agent Dynamism. Are the elements of the system (part of the organization, structure, interaction patterns) fixed or adaptable? Has the system learning capabilities?
If each agent performs a specific dedicated function (a specialist), the organization is called to be relatively fixed. If an application incorporates diverse functionalities (specialities) and has a high degree of complexity which necessitates clear function-to-agent mapping and clearly identifiable flows of control, then this type of organization is recommended. A fixed organization structure allows for surveillable load balancing of the limited resources such as communication bandwidth and computing power (in particular with a new or ill-understood application). When an application evolves and matures (and is better understood), these constraints could be more relaxed. Ultimate agent dynanism is the ability of reconfiguration or even self-design dependent of how the environment evolves or interacts with the application.

Agent Autonomy. How is control in the system distributed? To what extent are the elements autonomous?
Systems range from totally free groups (anarchy) to master-slave relations. The behaviour of free groups are difficult to predict and control. In safety-critical applications, agents will be controlled in order to exhibit a predictable and controllable behaviour. Agents may be controlled (i.e. their activation and interaction) by - possibly - a single control agent ("manager"), and/or by an operator (e.g. acting as a final authority) in a way depending on the current situation and operator preferences. Prime directive in safety-critical application is often that the human is always in command, so that agents will have minimum autonomy. However, some agents could have autonomy delegated from the human, especially in cases of high workload or routinely tasks.

Agent Resources. What resources are to which extent for whom available?
Resource availability in the system and limits of their utilization form one of the most crucial concerns for the designer. Whether the resources are ample or whether they are tightly limited affects the design and its effectiveness and can tilt the balance in favour of one design or another. Resources include computer power, time, memory, etc.

Agent Interactions. What type of interaction between the elements of the system are allowed? The range goes from very simple (e.g. neural networks), as well as uniform types of interaction to complex interactions.

Result Formation (Problem Solving Strategy). Does the system work by decomposing the problem into components (top-down) or by synthesizing existing elements?

For a more thorough analysis to reveal the characteristics of an application using DAI technology, [Dec89] could be consulted. A questionnaire is included for guidance in construction of a multi-agent based system architecture and identification of specific problem areas.

3 Functionality

This section discusses the main aspects of DPS in terms of functionality, methods and techniques. Sections 3.1 to 3.3 discuss the aspects along a number of typical phases in distributed problem solving. [Uma93, Smi81] divide DPS into four phases (Fig. 2):

- Problem decomposition.
- Sub-problem distribution.
- Sub-problem solution.
- Result synthesis.

The phases interact in the following way. On basis of a problem description, the problem to be solved is decomposed into a set of sub-problems that are individually solvable. These sub-problems are distributed among the agents taking into account a number of criteria such as their capabilities to solve that particular sub-problem and available resources. If the solutions of each sub-problem are available (possibly through cooperation among agents), these results are integrated to yield the solution of the overall problem.

The term "problem" is not the only key-word in distributed problem solving technology. In many applications, the word "task" plays a central role. A task is an activity performed by the the system, its user or both in harmony. In this respect, the phases of DPS become:

- Task decomposition.
- Task distribution.
- Task execution and coordination (cooperation).

As an example, consider the application Crew Assistant, an intelligent on-board system that assists the pilot or crew in performing the mission. The generic task of the crew is to perform the mission. This generic task can be decomposed in multiple sub-tasks to be performed by the crew. Some tasks need to be supported by the Crew Assistant. These tasks will be distributed among the different agents (or modules). Each agent will provide support to the crew on the task(s) allocated to him. Because most of the tasks are interdependent, agents have to interact and coordinate in order to support the crew in a consistent manner. Hence, task execution and coordination are intertwined, and therefore they are put together under the concept cooperation. (Sub)result-sharing and result synthesis are implicitly part of cooperation.

In the remaining, "(solving a) problem" and "(executing a) task" are used interchangeably.

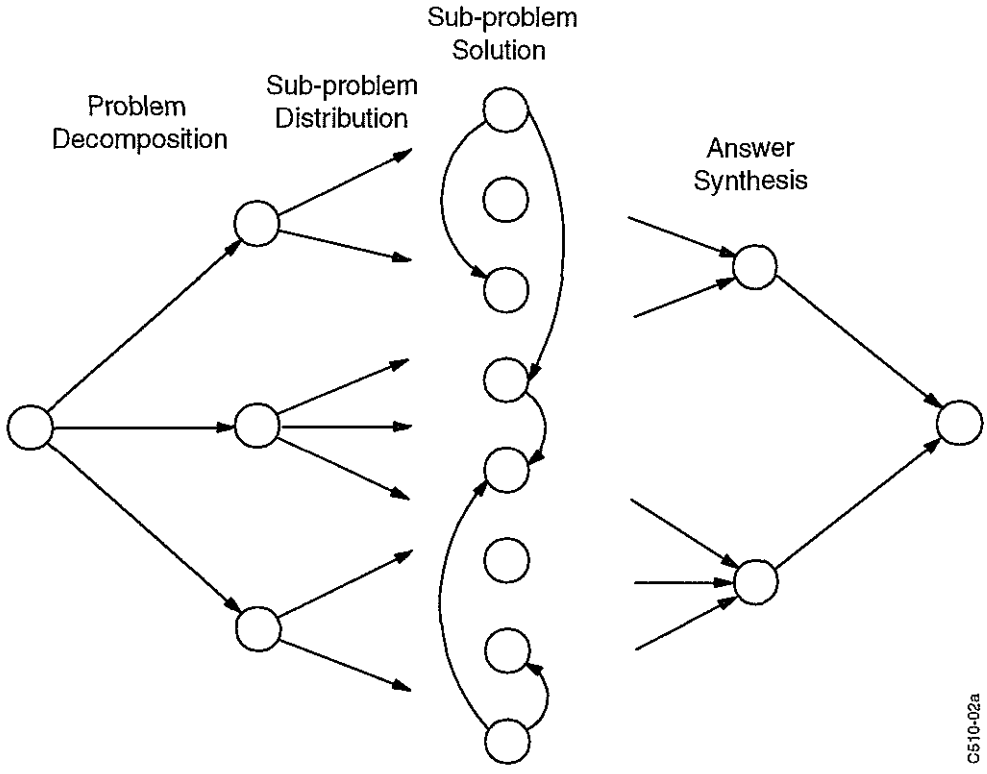


Fig. 2 Phases of distributed problem solving [Smi81]

To provide a framework for discussion of the basic functionality of DPS, sections 3.1-3 discuss each of the DPS phases and their associated methods and techniques:

- Decomposition.
- Distribution.
- Cooperation.

Further, section 3.4 discusses two architectures for DPS: the blackboard system, and the multi-agent system. Finally, section 3.5 will provide an overview of the main items discussed in this chapter.

3.1 Decomposition

Decomposition is the process of decomposing a task (problem) into a number of (or hierarchy of) sub-tasks (sub-problems) that are feasible to perform (solve).

3.1.1 Representation

Decomposition is based on the *description* of the task (problem) to be decomposed. This description is critical for decomposition, because it is the collection of attributes and descriptive

categories of tasks that provides a language for expressing inter-task and inter-agent dependencies. A representation is needed that makes formulation of the contents of a task possible (possibly consisting of a set of sub-tasks), and its boundaries and relations with other tasks.

This representation is developed by the system designer. In fact, the formulation (definition) of the tasks is also often done by the designer and hence is a priori known to the system. The representation of data, information, knowledge, problems and tasks are strongly related to the application domain. Therefore, these descriptions and together with the knowledge to reason with can be derived e.g. from domain experts by interviews or manuals. This makes for example formal task descriptions possible which can be included in a representation suitable for the system.

3.1.2 Dimensions

The basis for task decomposition is to find dependencies and logical groupings in problem tasks and knowledge. Dimensions of decomposition are [Bon88a, Gas92a]:

- Temporal (e.g. task i/o sequences).
- Knowledge, control and (input) data ("interest areas").
- Location.
- Abstraction (e.g. hierarchical levels of data).
- Functional/product.
- Resource minimization (e.g. minimizing task dependencies to reduce communication).
- Redundancy (e.g. overlapping tasks for reasons of reliability).

Depending on the application domain and available tools, some dimensions are emphasized more than others. The underlying conditions for this shading are basically the availability of agents to perform sub-tasks and minimization of costs of knowledge distribution (and hence communication) and resource distribution when assigning tasks to agents.

3.1.3 Decomposition and problems

Main problems encountered in the decomposition process are the presence of dependencies among sub-task decisions and actions of separate agents, and optimal use of resources, which may create conflicts with respect to high communication, incompatible actions and shared resources. If redundancy is required to improve reliability, these problems will get even more complex.

A number of methods are known to reduce or solve decomposition problems. These include:

- Pick tasks that are inherently decomposable, where the given representation of tasks contains its decomposition.

- Decomposition by the designer or programmer by built-in programmer-generated action descriptions (like the representation and formulation/definition of tasks, the process of decomposition is also often forethought by the designer rather than applying dynamic decomposition).
- Hierarchical planning by generating tasks that are goals to work on.
- Minimally connected graphs if the problem task can be described by a collection of interdependent elements.
- Subtask aggregation by composition of operators to fit the requirements of subparts of a larger task.

For time/safety-critical applications, decomposition should be performed on basis of a predefined task hierarchy. This hierarchy should be based on the *dimensions* of decomposition (see section 3.1.2) in order to obtain an efficient, predictable distribution of tasks among the available agents and have an optimal use of resources. Minimal resource-sharing and minimal task dependencies will mean minimal coordination among the agents (enhancing real-time performance).

3.2 Distribution

The problem of distributing tasks among agents is the problem of assigning responsibility for a particular activity. Task distribution is a meta-problem that may be addressed statically by the designer or may be done dynamically by a collection of agents themselves. Two main aspects of task distribution are discussed:

- Task allocation.
- Resource allocation.

3.2.1 Task allocation

Task allocation is concerned with which agent should get which tasks. There are several methods and guidelines for allocating tasks to agents (dynamically or statically by the designer):

- *Bottleneck avoidance.* Task allocation should avoid bottlenecks by overloading a particular unique or critical agent or resource. This means that the set of agents should be balanced with respect to the set of tasks to be performed.
- *Fit to specification.* Tasks should be allocated to those agents that provide the best fit to the task specification.
- *Knowledge dependency.* Task coordination should be left to the agent with the most global view. After decomposition and allocation of the tasks are allocated to the specialized agents, a global view might be lacking in order to keep track of task interdependencies. A kind of control task (meta-task) assigned to an agent having a global view (i.e. knowledge) of the system and its agents might be necessary.

- *Overlap in roles*. For reasons of flexibility, reliability and coherence, a task can be worked on by more than one agent. However, to cope with this redundancy and to avoid conflicts, the agent's responsibilities in a partial solution space must be known.
- *Uncertainty avoidance and reliability*. Tasks whose results or completion (in time) are uncertain should be allocated redundantly to reduce the uncertainty and improve reliability. To manage redundant allocation, conflict resolution schemes should be available.
- *Resource consumption.*
  Tasks should be allocated in such way to minimize use of resources (see section 3.2.2).
- *Urgency*. Urgent tasks should be allocated to agents that can directly perform them. This implies that redundant allocation of tasks (i.e. there are two or more agents that can execute a specific task) should be possible.

The following typical mechanisms for making task allocation decisions are available:
- *Market mechanisms*, wherein available tasks are matched with available agents by generalized agreement and possibly mutual selection such as Contract Net [Smi88].
- *Multi-agent planning*, wherein a planner or collection of planners can combine the work of task decomposition and task allocation by treating agents as specialized resources or objects that interact and depend on one another, such as partial global planning [Dur87] (see also [Mar92]).
- *Organizational roles*, that are predetermined and slowly changing policies;.
- *Recursive allocation*, by letting agents that are handling problems do the work of allocating subproblems [Dav83, Wes81].
- *Voting*, where a set of agents vote to let one agent do a task [Ste86b].

Most of the discussed methods and mechanisms take time to allocate the tasks to agents. Time is critical in a real-time application. If there is enough time, a more deliberate method may be used, but if time is scarce, the system should fall back to a quick pre-defined method based on pre-determined organization roles. In this respect, and taking into account that task decomposition is bestly to be based on a pre-defined task hierarchy and that the different agents are clearly related to and specifically designed to perform these tasks (agents' roles are fixed), task allocation is recommended to be based on a pre-defined mapping of tasks to agents. So, a task allocation scheme should be defined by the designer and embedded in the system architecture and structure of the agents of the application. This A clear mapping defining the allocation of tasks to agents will imply a modular architecture that in particular addresses the aspects mentioned at the the beginning of this section.

3.2.2 Resource allocation

An important aspect of task distribution is *resource allocation*. Resources are the products that are consumed or used to accomplish problem-solving work. Task decomposition and distribution should take the limited availability of resources into account. Resource-bounded reasoning is important in any real system, and recent research has started to address trade-offs in resource allocation and real-time performance such as using a reasoned approach to reduce search complexity called approximate reasoning [Les89] (see section 3.4).

The following resources are of most importance for complex critical applications:

- Computational resources (processor power, number of processors).
- Communication bandwidth (for interaction between agents).
- Memory (for storing current situation and knowledge).
- Completion time (when response time is critical).
- Sensory systems (different sensors for different purposes).
- Effectors, actuators (displays, tentacles, etc.).
- Cognitive limits (bounded rationality) of operator (prevent data overwhelming or saturation).

Limited computer hardware makes the availability of the first three resources scarce which endangers completion time and real-time performance. Conflicts must be solved if different agents make use of the same sensors, effectors or actuators which themselves may be expendable. Last but not least, the limited cognitive capabilities of an operator have to be taken into account, and conflicts may arise if there is more information to be presented than the operator can absorb (e.g. which agent may present first or when its information) [Ger87][2)].

Allocation of resources is done at run-time and depends on the task distribution and current availability of resources. Major criterion for resource allocation is that the most pressing and critical activities must be done first. This requires a task prioritization scheme that depends on the situation. Furthermore, to come to a balanced allocation (minimal or optimal use of resources), some predictions on future system performance has to be done. These predictions are uncertain or are infeasible at all, in particular for applications operating in or interacting with a highly dynamic environment. Progressive reasoning may be a solution to this problem. This type of reasoning will gradually allocate or consume resources during problem solving. It will always provide a problem solution, but the maturity or detail depends on the amount of resources (e.g. available response time).

---

2) Basically, the human has available for communication or interface channels with a system s/he operates [Ger87]: vision, audio, speech, and tactile. In principle, all four channels could be used independently, but in practice, human motor coordination and cognitive limits will restrict simultaneous use of more than two channels. This should be taken into account as a constraint (a cognitive limit) on the use of resources.

3.3 Cooperation

Cooperation among agents is necessary because of the existence of interdependencies among tasks distributed across different agents. Agents need to cooperate on basis of these dependencies in order to accomplish the individually assigned tasks or to reach some common goal. Cooperation is discussed along three main aspects:

- Interaction.
- Coordination.
- Coherence.

These aspects relate to each other in the following way. *Coordination* is characterized as patterns of interaction (activity) among agents. The definition of coordination is based on the concept of interaction. *Interaction* is some type of collective action in a distributed problem solving system, wherein each agent takes an action or makes a decision that has been influenced by the presence or knowledge of another agent. This influence can be realized through a communication channel by exchanging messages with other agents, or through the real world by actions of other agents. Finally, coherence says something about coordination in the total DPS system. *Coherence* refers to how well the system behaves as a unit, i.e. how well-coordinated the system is.

3.3.1 Interaction

Interaction critically depends on the employed communication primitives (representation, communication language, protocol) and the agents' model of one another. Important aspects of interaction include:

- Among whom does the interaction take place.
- When does the interaction take place (temporal and causal relationships among agents).
- What is the contents (e.g. results to be shared).
- How is interaction accomplished (e.g. what processes are involved or what resources are utilized).
- What interaction primitives are used (e.g. protocols, message passing, shared memory).
- Why does interaction take place (agent's goals).
- What is the basis of commonality to interact (e.g. shared interpretative context).

For the design of a modular-time application, it is important that the designer asks himself these questions, both at task level and agent level (after task distribution). This will reveal the internal interfaces of the system and hence will provide a good basis for a modular architecture. A solid interaction concept will mean a good basis for coordination and will enable coherent behaviour.

Several levels of interaction exist between agents. [Dem90] identifies three kinds of information exchange (result-sharing) based on an agent model (see Fig. 3) that takes into account that its

knowledge about the world and other agents is incomplete, uncertain and partly erroneous. The kinds of information exchange are:

- *Knowledge*, due to differences and incompleteness in perceiving the environment (sensor or interpreted data), agents may have complementary, overlapping or conflicting descriptions of a shared situation.
- *Possible solutions*, the exchange of possible solutions (e.g. action plans) arises when two or more agents have to agree on a common solution or at least a non-conflicting solution in order to satisfy individual goals or system goals.
- *Choice*, given a set of common possible solutions, one has to agree on a common choice when cooperation is needed.

Based on these kinds of information exchange, three types of interaction exist as illustrated by figures 3a-c:

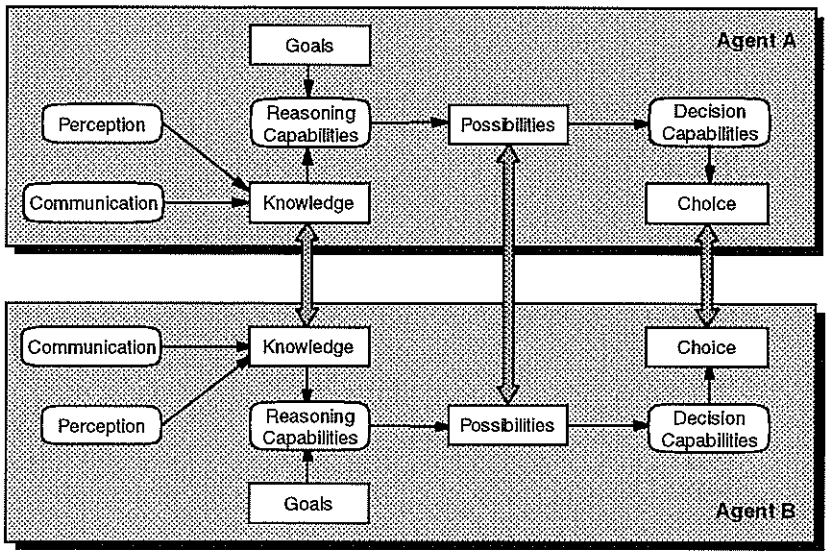
- Strong interaction between decision capabilities.
- Medium interaction between reasoning capabilities.
- Weak interaction between perceiving capabilities.

3.3.2 Coordination

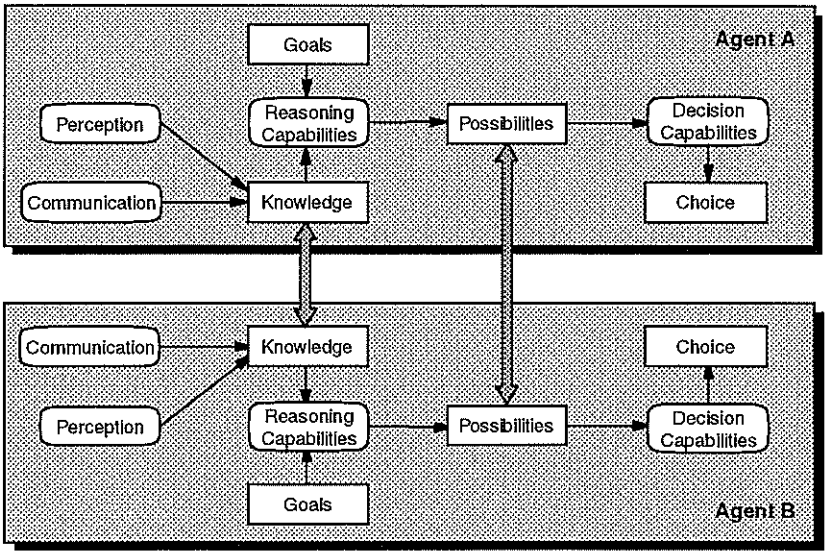
Coordination is concerned with the patterns of interactions that make agents work together. It is the ability to combine their activities to achieve a common purpose [Lan94]. This section discusses a number of methods for coordination.

Organization. An organizational structure is the pattern of information and control relationships that exist between agents, and the distribution of problem-solving capabilities among the agents [Dur89]. An organization can provide a framework of constraints and expectations about the behaviour (roles) of agents that focuses the decision making, processing and communication resources, and action of particular agents that are likely to lead to effective network performance. Different kind of organizations exist:

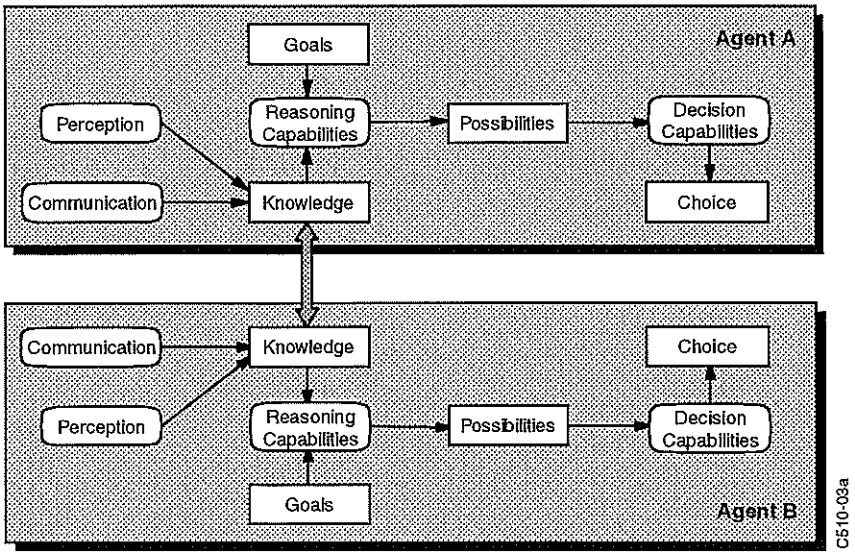
- *Centralized* and *hierarchical* organization, that typically associates greater control to a more global viewpoint, in which agents with more global information guide agents with less global information as decision-making data flow "upward" in progressively more abstracted forms, and control data flow "downward" (a centralized organization typically contains a management agent such as [Ger87] which discusses an expert system management system).
- *Authority structure*, in which agents have authority over others because they have more accurate views of a situation than others, with as most-extreme structure the master-slave relation.



(a) Strong interaction between decision capabilities.



(b) Medium interaction between reasoning capabilities.



(c) Weak interaction between perceiving capabilities.

Fig. 3 Types of interaction [Dem90]

- *Market-like* organization, in which agents can dynamically negotiate for assignment and execution of tasks and how to cooperate effectively (e.g. Contract Net [Smi88]).
- *Community*, where an organization is constructed as a set of locally interpreted rules of behaviour rather than as an externally defined structure. An extreme case is an anarchy with no rules at all.

In order to oversee the complexity inherent to envisaged critical applications, and enhance real-time performance, a well-defined, pre-designed organization is desired in order state clearly what each agent has to do and how (i.e. the agent's role). This rules out market-like organizations. These type of organizations have dynamic negotiation as key strategy and assume well-defined task hierarchies that can be dynamically decomposed into nearly independent sub-tasks, which is likely not be the case with many applications.

Community-like organizations are preferred above centralized organizations, because of aspects such as reliability (the network performance should not rely on one agent), modularity, limited computation (the problem of coordinating many agents is computational intractable for a single coordinator), and limited communication (a single coordinator could be a communication bottleneck and could be overwhelmed with information from other agents) [Dur89].

Localization. The degree of localization of knowledge, responsibilities, control and capabilities affect the way of coordination. It concerns integration of reasoning about other agents' actions and beliefs with reasoning about local problem solving, so that coordination decisions are part of local decisions rather than a separate layer above local problem solving. It concentrates on how to build agents that can decide for themselves how and when to coordinate, rather than having a specific coordination approach imposed on them. Methods for improving localization are:

- *Specialization*, which improves performance by reducing and focusing responsibilities of an agent, and hence reducing its local decision-making overhead (direct decisions are possible that otherwise involved multiple steps) [Dur87].
- *Dependency reduction*, which improves coordination by reduction of local dependency among agents so that there is less possibility for harmful interaction (conflicts) and correspondingly lower computation or communication overhead.
- *Local capabilities*, increased local capabilities (possibly redundantly allocated to multiple agents) will improve coordination by more local problem-solving knowledge, more internal control and greater resources and - hence - by making possible to evaluate each of its decisions on how it will affect network problem solving (e.g. local planning [Dur87]). Increase of local capabilities will reduce communication, but will increase local overhead.

All localization strategies are applicable to real-time applications. Specialization and dependency reduction depend on the set of tasks and their dependencies and the allocation of tasks to agents. Increase of capabilities of an agent (or module) is mainly realized by providing it with sufficient problem-solving knowledge about the agent-specific problem domain.

Planning. Coordination can be performed and improved by aligning behaviour of agents toward common goals or making use of common resources through planning of activities. Interaction on plans have to take place to resolve incompatible states, order of steps, use of resources, and also to perform task distribution. In this way, activities can be synchronized and conflicts can be avoided before actual execution. Planning activities can be performed by a single agent (centralized planning), or can be divided up among multiple agents (distributed planning [Mar92, Dur87]).

Further methods to improve coordination are:
- Increase *contextual awareness* of agents so that they can make better decisions.
- *Communication management* to be aware what, how and when to communicate in which relevance, timeliness, and completeness are key items [Dur87, Dur89].
- *Resource management* in order to avoid conflicts.
- *Data abstraction* and *meta-level information* about the problem domain and inter-agent communication respectively, which directs coordination.

3.3.3 Coherence

Coherence is the extent to which agents behave as a single unit in harmony, exhibiting coordination and consistency (which is defined as the degree of agreement between agent's conclusions) [Lan94].

Coherence can be evaluated along several dimensions of system behaviour [Gas89]:
- *Solution quality*, the system's ability to reach satisfactory solutions of a certain quality in the presence of uncertainty in data, knowledge, control, processing algorithms and communication.
- *Efficiency (responsiveness)*, the system's overall efficiency in achieving some end and to respond to events within the required time limits through efficient use of communication and processing resources.
- *Clarity*, the conceptual clarity of the system's actions, and the usefulness of its representation understandable by an outside system observer and appropriate for self-representation of the system (with respect to communication. organization, performance analysis, etc); and
- *Graceful degradation (reliability)*, the degree the performance of the system degrades in the presence of failure (agent or communication) or uncertainty.

The primary difficulty in establishing coherence is the attempt to achieve it without centralized (coordination) control or viewpoints, but with distributed control and distributed (possibly different) viewpoints which are locally achieved within each agent. [Les87] argues that obtaining coherent behaviour in a DPS system requires the achievement of three conditions:

- *Coverage*, each necessary portion of the overall problem must be included in the activities of at least one agent.
- *Connectivity*, agents must interact in a manner that permits the covering activities to be developed and integrated into an overall solution, and agents must interact to avoid inconsistencies.
- *Capability*, coverage and connectivity must be achievable within the communication and computation-resource limitations of the DPS system.

The system design of a complex real-time application based on DPS technology should take these conditions as system design attributes into account. They are the basis for the system architecture. The complexity of the application makes the realisation of coverage and connectivity (which relate to decomposition, distribution and cooperation) a non-trivial task. With respect to the last condition ("capability"), the application of DPS in a large, complex, and real-time system assumes the availability of extensive computer hardware resources based on multi-processor technology. If this assumption is not met, communication and computation resources will likely not allow for application of DPS on a large scale.

3.4 Architectural approaches to DPS

As was already indicated in section 2.1, the research community recognizes two relatively well-established architectural approaches to DPS:

- Blackboard systems.
- Multi-agent systems.

Both type of systems consist of multiple agents, but they differ in structure at both global architecture level and agent level (see Fig. 1). Both will be discussed and reference is made to the methods and techniques discussed in previous sections.

3.4.1 Blackboard systems

A blackboard system [Eri85, Nii86] relies on a conceptual, high-level organization of information and knowledge needed to solve a problem, and a general prescription for the dynamic control and use of knowledge for incremental, opportunistic problem solving. The blackboard system consists of three components:

- *Knowledge sources*. The knowledge needed to solve the problem, is partitioned in knowledge sources ('agents'), which are kept separate and independent. Each of the knowledge sources

is an expert in some area, and may find a hypothesis (on the blackboard) it can work on, solve that hypothesis, create new hypotheses, and modify other existing hypotheses. A knowledge source is not considered as an 'intelligent' agent in the sense that it has no local control or data/information storage capabilities.

- *Blackboard data structure.* The problem-solving state data are kept in a global data store, the blackboard. Knowledge sources produce changes to the blackboard, which lead incrementally to a solution to the problem. Communication and interaction among the knowledge sources takes place solely through the blackboard.
- *Control.* Problems addressed in control are knowledge source activation (which knowledge sources to apply when and to what part of the blackboard) and communication among knowledge sources in order to speed up the problem-solving process (in restricted cases) or guarantee convergence to a solution.

An overview of the blackboard architecture is given in Figure 4. The control data rectangle represents the data needed for the control mechanism to function (control knowledge) and is not part of the knowledge sources needed for the problem (domain knowledge).

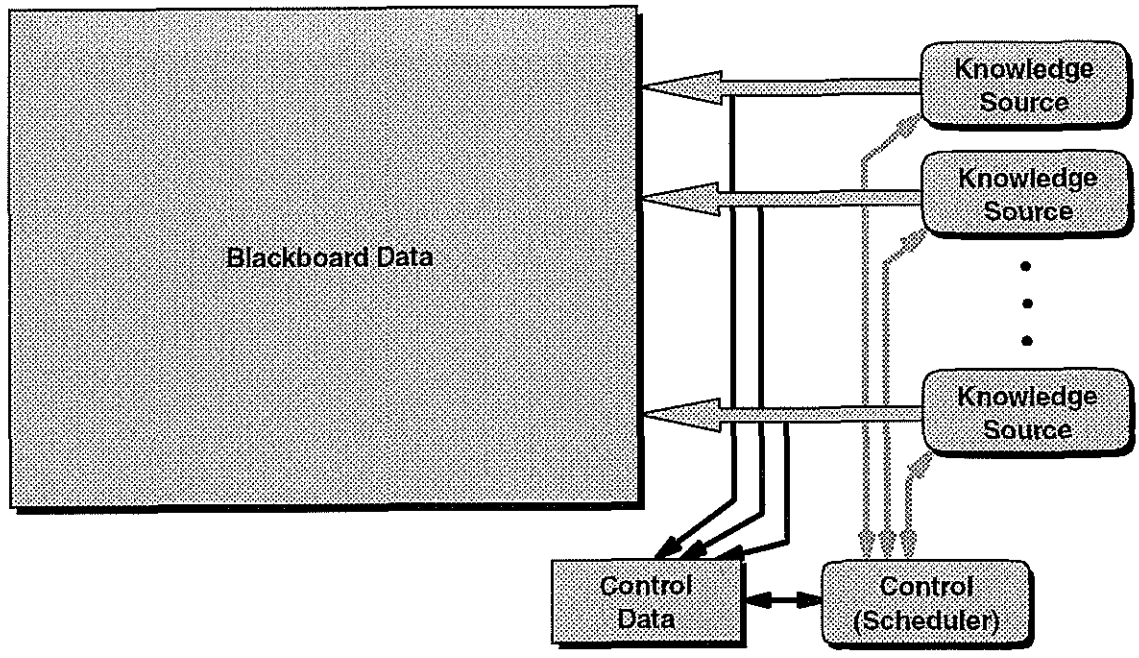


Fig. 4 The blackboard system architecture

A particular reasoning behaviour is associated with blackboard systems: the solution to a problem is built one step at a time. At each control cycle any type of reasoning step (forward

chaining, backward chaining, etc.) can be used. The part of the emerging solution to work on next, can also be selected at each control cycle (relates to the control strategy: focus of attention). As a result, the selection and the application of knowledge sources are dynamic and opportunistic rather than fixed and preprogrammed.

Although the blackboard model was originally conceived as a model in which the knowledge sources were to be executed in parallel, practical implementations used to be strictly sequential. However, the increasing availability of real parallel machines can reverse this trend. Based on this information, three blackboard multiprocessor architectures can be distinguished [Cor89b]:

- The *shared-memory* blackboard allows each processor to directly access one central blackboard.
- With the *distributed* blackboard, each processor accesses a separate local blackboard. A communication channel is needed to exchange information between the blackboards (e.g. Distributed Vehicle Monitoring Testbed DVMT [Les83] and Hearsay-II [Erm80]).
- In the blackboard *server* approach, just one processor can access one central blackboard.

Actually, these approaches are intermediate forms from the conventional blackboard model to multi-agent systems. The blackboard server approach is even a special type of multi-agent architecture in which the information of an agent is stored in a private blackboard and only communication of these blackboard data is allowed (e.g. through message passing). The local blackboard can only be accessed by the agent owning it, and communication of blackboard data is handled by that agent as opposed to the distributed blackboard approach where direct read/write access to a local blackboard is possible by another agent.

In the light of the aspects of DPS as discussed in sections 3.1 to 3.3 the following can be remarked. A (conventional) blackboard system results from a top-down design. It involves how to divide a particular *single* problem or task among a number of knowledge sources and how to design a suitable (possibly hierarchical) blackboard representation for the problem to be solved or task to be done. This decomposition activity is an integral part of the design of a blackboard system. Therefore, no *dynamic* task decomposition into sub-tasks is involved.

Task distribution is also left as a design problem. Each knowledge source fulfils a specific sub-task or solves a specific sub-problem. It provides a contribution to the overall task or problem. In this respect, decomposition and distribution are an intertwined problem for the designer.

With respect to cooperation, the following can be remarked. The problem or task is divided among a number of knowledge sources, that cooperate at the level of dividing and sharing knowledge about the problem and about developing a solution. The interaction and coordination

strategies of knowledge sources through the backboard structure are an integral part of the design as well. In many blackboard systems, coordination is implemented by a centralized scheduler. It is often the case that this scheduler is the bottleneck of a blackboard system with respect to complexity and performance. In fact, this problem motivated the design of distributed blackboard and blackboard server systems, and subsequently multi-agent systems.

Considering the aspects of coordination as discussed in section 3.3.2, the following remarks can be made in particular:

- Organization is mainly realized by hierarchical representation of the blackboard and designing knowledge sources that transfer (interpret) data between the hierarchical levels. Knowledge sources at the lower level in the hierarchy are mainly data-driven and work on problems and data with few abstraction (e.g. raw sensor data) in order to prepare information for higher levels of reasoning, whereas high-level knowledge sources are mainly goal-driven and work with symbolic *abstract* (meta-level) data in order to work towards a solution of the problem or the achievement of the task.
- Localization is minor and planning, communication and resource management are part of the central scheduler. In this respect, control is completely centralized, which makes the blackboard system vulnerable to failure.

With respect to highly complex critical applications, the blackboard model is not suitable to act as the basic architecture, in particular because of its centralized control and data management. However, specific problems or tasks to be performed by an agent (or module) may be suitable for a blackboard system implementation. In particular those problems or tasks that are concerned with a large solution space, noisy and unreliable input data, a variety of input data and a need to integrate diverse information, the need for many independent or semi-independent pieces of knowledge to cooperate in forming a solution, the need to use multiple reasoning methods and/or the need for an evolutionary (incremental) solution [Jag89].

3.4.2 Multi-agent systems

Unlike in blackboard system, data, information, and control in multi-agent systems, are distributed among agents. Each agent has beside the knowledge its private data and information and its own control cycle.

The general contrast with blackboard systems is that in a multi-agent system the agents are autonomous, potentially pre-existing, and typically heterogeneous. Multi-agent systems are concerned with coordinating intelligent behaviour among a collection of intelligent agents: how to coordinate their knowledge, goals, skills, and plans jointly to take action or to solve problems. They work toward a single global goal, or toward separate individual goals that interact. They must share, like blackboard systems, knowledge about problems and solutions, but they must

also reason about the processes of coordination among the agents. A multi-agent system can be viewed as a bottom-up designed system where agents are designed first, and a solution strategy for a given problem is specified later.

The aspects discussed in the sections 3.1-3 could in many cases be dynamically performed by the system itself, although the designer may decide to develop some built-in fixed strategies for decomposition, distribution and cooperation to control complexity and performance.

With respect to task decomposition and distribution, they can be dynamically performed by the system rather than pre-specified by the designer. The ability of the system to perform dynamic decomposition and distribution heavily depends on the problem domain and system design. For example, agents having a common or similar design (e.g. knowledge and data representation, reasoning schemes, functionality, etc.), will make decomposition and distribution given the available resources and agents' abilities much easier. However, if agents are very heterogeneous and have different functions (specializations), decomposition and distribution will be more constraint towards the functional capabilities of the agents. In case of an a priori known agent organization, fixed decomposition and distribution schemes in the system design are often desirable in order to avoid unnecessary overhead during run-time due to negotiation.

The aspect of cooperation in multi-agent systems has been mostly often addressed. [Les81] provides a good introduction. The need for coordination and achieving coherence in multi-agent systems lies in the overlapping or potentially conflictuous activities of agents. Achieving coherence in a multi-agent system is additionally hindered by the fact that each agent will have a certain view of the total system and environment that might differ from other agents due to the uncertainty and limitation of (perceived) data, knowledge, control, and processing algorithms. The resulting limitations on effective information processing and control capabilities determine the *bounded rationality* of an agent. The rational bounds of an agent are characterised by the scope of its local decisions (control bounds), the information that is used to update these decisions (interpretation bounds) and the updating process (bounds on the nature of decision making). [Les81] discusses a number of design methods to increase the rational bounds of an agent based on these characteristics - and hence decrease agent's uncertainty - in order to achieve coherence. Beside application of knowledge-based system related methods (e.g. representation, uncertainty reasoning, search control, planning, incremental reasoning), [Les81] suggests to relax self-directed control and introduce more explicit, externally-directed control mechanisms. This means that organizational structure of a multi-agent system plays an important role as a kind of explicit planning of control among agents. Note that bounded rationality can also be increased by considering the aspects of coordination in section 3.3.2.

Due to the limited rational bounds of agents, conflicts will arise between the beliefs and decisions of agents. Conflict detection and resolution are important aspects of cooperation in multi-agent systems. The set of potential conflicts should be kept to a minimum and, therefore, should be a design goal itself. However, it is not likely to obviate all potential conflicts in the design of the system, calling the need for methods to detect and solve emerging conflicts dynamically. Methods for conflict resolution include total ignorance (do nothing), conflict avoidance (e.g. through common and consistent views of the world, uncertainty reduction, etc.), enforcement (e.g. through arbitration by a single agent), negotiation, etc. Much of the research on conflict handling focuses on negotiation. Conflict resolution strategies and techniques through negotiation are discussed in [Syc89], [Adl89a], and [Pol93].

Multi-agent systems inherit the nice properties of DPS, as they were discussed in section 2.2. However, the potentially conflictuous behaviour and extensive coordination among agents require a deliberate design. A critical application must be highly reliable and must perform in real-time which ties task decomposition, distribution and cooperation to strict time limits. The designer should consider whether a dynamic approach to these aspects will be taken with the necessary overhead or a more static approach. It is recommended to consider the latter as a viable option in which the designer will fix certain strategies (e.g. strict organization with fixed agent roles, pre-programmed decomposition and distribution schemes, etc.) to reduce overhead (e.g. extensive negotiation) and complexity.

3.5 Conclusion

This section discussed the main aspects of DPS in terms of functionality, methods, and techniques.

From a functional point of view, relating DPS to complex, modular critical applications as discussed in this section shows that blackboard systems and multi-agent systems are relatively made-to-measure technologies. These technologies can be applied to both module and system level. Feasible application of blackboard systems mostly implies feasible application of multi-agent systems, because multi-agent systems incorporate the functionality of blackboard systems, but distribute also data and control.

Although section 2.2 suggests a reduction of complexity by having a system decomposed in multiple cooperative agents, the overall complexity of applying DPS to any critical system should not be underestimated. To get the bottom of the cooperative aspect among agents is far from trivial. The way of coordination and achieving coherence remains complex and needs deliberate system design. In order to control complexity as well as to achieve the required performance, decomposition, distribution and cooperation strategies should not be too flexible. In fact, it is argued to embed fixed strategies in the design of the system by the designer himself,

rather than letting the system itself dynamically apply strategies introducing overhead and possibly incoherent (non-convergent or non-predictable) behaviour.

To avoid this harmful system behaviour, the following measures can be taken.

- Apply decomposition on basis of a formally prescribed task hierarchy that considers a number of criteria in order to obtain an efficient distribution of tasks among the agents and to have an optimal use of resources (section 3.1).
- Distribute tasks among agents on basis of the resulting decomposition and easy to calculate (possibly a priori known) task prioritization schemes.
- Base the agent structure on the model as discussed in section 3.3.1 and apply strong interaction in order to avoid conflicts between agents and achieve coherent behaviour.
- Design a fixed community-like organization of agents with strict rules of behaviour. This inflexibility may be loosened during the evolution of the application.
- Make extensive hardware resources available based on multi-processor technology (in fact, this is a requirement for application of DPS on large scale).

4 Evaluation

This chapter provides an evaluation of DPS technology along the following criteria to be considered important for complex modular critical applications:

- Reliability.
- Performance.
- Modularity.
- Integrability with other technologies.
- System engineering (methodology and user interfaces).
- Maturity and next generation.

4.1 Reliability

Reliability can be defined as the probability that the system performs its assigned functions under specified environmental conditions for a given period of time [Rod93]. Reliability is one of the potential benefits gained from the application of multi-agent systems. Reliability is less in blackboard systems, because of the centralized control and blackboard data structure. Multi-agent systems are considered as potentially more reliable than conventional (monolithic) systems through the application of redundancy, cross-checking, and triangulation of results.

The remaining sections discuss how reliability can be achieved, which aspects could endanger reliability, and the last section provides concluding remarks on reliability.

4.1.1 Achievement of reliability

Reliability in multi-agent systems is achieved by the following characteristics.

Modularity. Multi-agent systems have a high degree of modularity. The actual processing is encapsulated in the agent itself, i.e. the internal processing and data structures of the agent are hidden from the other agents. No software links will be made to internal agent structures; exchange of data takes only place through well-defined interfaces and employing communication primitives such as message passing or shared memory. The modular design of a multi-agent system consisting of relatively simple, cooperative agents makes a complex system such as Crew Assistant comprehensible, surveillable, and maintainable, and hence potentially more reliable.

Redundancy. System capabilities can be redundantly allocated to agents. This redundancy and associated commonality among agents (i.e. overlap in the roles of the agents) provide flexibility in case of agent failure. It allows for tasks to be taken over by other agents or to be redundantly allocated from the beginning. This prevents the system from complete failure, but allows the system to perform at a degraded but reasonable level of performance. Optimal reliability from

a single agent's point of view is achieved if this agent can still perform its task, possibly in some degraded manner, if its external agents do not respond.

Integration of results and reduction of uncertainty. Agents generating common results, but obtained from different viewpoints (e.g. different sensors), increase reliability of and reduce data uncertainty in the system by cross-checking and triangulation. In this way, noisy, unreliable and uncertain data can be dealt with.

Multi-processor hardware. Multi-agent systems are perfectly suited to run on a network of multiple processors where each agent (or set of agents) runs on a private processor. This allows for parallelism at both software and hardware level. It allows for reliability, with possibly some performance degradation, in cases of both software or hardware failure.

4.1.2 Negative effects on reliability

Multi-agent systems are potentially more reliable than conventional, centralized systems, but on the other side, multi-agent systems might have negative effects on reliability which should be taken into account.

Non-determinism. Multi-agent systems have a potentially high degree of non-determinism. The behaviour of agents in complex critical applications is non-deterministic, because it is expected that knowledge-based system technology will be used and input data will be uncertain and dynamic. In addition, at system level cooperation is non-deterministic because of its asynchronity. This non-determinism has as consequence that the line of reasoning of the system will not be fully traceable or reproduced and hence can not be fully verified or validated. The non-traceability feature allows for forward error recovery only.
[Lan94] truly remarks that the system's reaction to unexpected events should be both predictable and reliable if it is to gain acceptance by a user community. Therefore, it is of high importance to keep non-deterministic behaviour to a minimum. This can be reduced at the cost of flexibility by incorporation of static strategies in the design of the system such as prescribed decomposition, distribution and cooperation methods and techniques. This fixation will also have a positive effect on managing complexity and achieving coherence, because it makes the system and its coordination patterns surveillable and predictable.
Non-deterministic behaviour can also be reduced by introduction of a strict data and control flow mechanism, likely to be centralized or hierarchical. Therefore, blackboard systems have a lower degree of non-determinism, because of centralization of control and data which makes the system more predictable (but also more vulnerable: complete system failure if the centralized controller fails).

Performance. Reliability is partly realized by redundancy of agent capabilities and integration of results. The underlying flexibility of agents plays a key role. This flexibility is at the cost of performance, because it requires extensive interaction among agents to allocate tasks (e.g. through negotiation), coordinate task execution and integrate results (including conflict resolution). Reliability is a very important issue in critical applications, but real-time performance even more. A critical application that ignores response time constraints is useless.

Testability. A number of features make multi-agent systems difficult to test [Avo92]:
- Many loci of control (simultaneous intervention is difficult).
- Communication delays (determining the system's state at a given time is difficult).
- Non-determinism (reproducibility is difficult).
- System monitoring alters behaviour (stopping or slowing down one process alter behaviour of the entire system).
- Large amounts of data (magnified in DPS systems that are often large).

Safety. The discussion above provides an indication of the problems that could emerge in the process of *verification, validation and certification* and associated safety as e.g. required in the aerospace domain [Ste86a]. These are issues that have not been explicitly addressed within the DPS community yet. The main problem with respect to verification, validation, and certification of this technology is how to deal with non-determinism. This especially holds for multi-agent systems. Non-determinism must be kept to an absolute minimum in order to gain user acceptance and - at the end - certification of an application. As indicated above, methods exist to reduce non-determinism.

4.1.3 Conclusion

It can be concluded that the application of DPS technology will increase reliability (and safety) of a system if non-determinism is kept to an absolute minimum. Total safety is only guaranteed if the following conditions are satisfied:
- The system's task is to *support* the user (human operator).
- The user will always be in command as final *authority*.
- Delegated autonomous operation may only be considered for simple, routinely tasks that ensures or approximates deterministic and predictable agent behaviour.

If these conditions apply, DPS technology will contribute to a higher reliability and overall safety.

4.2 Performance

This section discusses real-time behaviour of DPS technology. Section 4.2.1 provides an indication of performance requirements for a real-time application that must be handled by DPS

technology. Section 4.2.2 discusses a number of methods and techniques where a real-time application can benefit from. Section 4.2.3 provides some concluding remarks.

4.2.1 Performance requirements

A real-time (critical) application shall have guaranteed response times in an often highly dynamic environment. This makes real-time performance a critical factor in user acceptance. A real-time application shall deal at least with the following real-time operational requirements, i.e. the system shall be capable of handling [Lan94]:

- Asynchronous and unpredictable events.
- Dynamically changing data during problem solving (non-monotonicity, focus of control).
- Time constraints and the trade-off with the response quality (time-constrained reasoning).
- Reasoning about events in both space and time (time-stamped data: creation-time, validity time; temporal model of the system [Rod94], temporal and spatial reasoning).
- Uncertainty and sensor data (uncertainty reasoning, believe revision).
- Continuous operation (e.g. history management, garbage collection).

The capabilities to handle these requirements are constrained by the response time. With respect to real-time performance, three types of response (not exclusive) are identified [Kui94]:

- *Fast.* The ability of a system to compute a response fast. This is rather vague since the concept of fast cannot stand alone: it has to be compared with the response times of other systems.
- *Hard real-time.* The ability of a system to guarantee a response after a fixed time (defined before run-time) has elapsed. This viewpoint is especially important in time-critical situations where it has to be one hundred percent certain that a response is given before the deadline (i.e. the maximum response time) has elapsed.
- *Any-time.* The ability of a system to produce a response at any-time: a response can be given whenever is needed (but quality of response is likely to increase after more time has elapsed).

The inherent parallelism of multi-agent systems and the ability to run on multi-processor hardware through natural distribution of agents among the available processors allow for *fast* response. Increased fast response is also obtained in multi-agent systems where agents are co-located and make use of shared memory rather than message passing as communication primitive [Dec87].

With respect to blackboard systems, they are recognized as being slower due to the centralized approach of control and data (see section 3.4.1). For example, [Rau89] provides an overview of real-time performance problems of blackboard systems, attempts to solve them, but concludes that the resulting prototype still suffers from real-time performance problems.

In any way, fast response is not enough for time critical applications. Complex real-time applications in a highly dynamic, complex or demanding environment have to be surely categorized as being a hard real-time system, with preferably any-time response features. As [Hay94] says, the *utility* of a system's behaviour is a function of the *criticality* of the events to which it responds and the *value* of its response to them. This value contains a basic trade-off between response *quality* (correctness, completeness, precision, etc. of the response) and response *latency* (the delay between occurrence of the event and the response). In general, the faster the response, the lower the quality of that response. If latency is subject to a hard deadline, then violation reduces response value directly to zero. In many situations, hard deadlines will be the case in real-time safety-critical systems.

4.2.2 Real-time methods and techniques

If DPS technology is to be applied in complex real-time systems, it should incorporate techniques that cope with hard real-time and any-time requirements (including those mentioned in the beginning of section 4.2.1), deal with the basic trade-off of quality and time, and consider predictability of responsiveness and upper-bounds on response times, under the assumption that resources (e.g. computation, communication) are limited (a very practical assumption). In literature, a number of promising methods and techniques appear.

[Lan94] presents a planning method called PAO^* for meeting event deadline specifications at both architectural and agent level. It is based on decomposition and allocation of deadlines to agents given the tasks to be done and the available resources. The technique assumes the existence of redundancy in the multi-agent system (i.e. a task can be performed by multiple agents). The planning technique addresses all aspects mentioned in the beginning of section 4.2.1. The technique is based on a trade-off between quality and time performed by the agents which has its effect on the coherence of the total system.

[Hol94, Rod94] introduce *dynamic notice boards* as basic element for real-time operation using blackboard system technology. Each agent is assigned a notice board to communicate. It addresses synchronization issues and temporality of data. It has been implemented in DENIS - a Dynamic Embedded Noticeboard Information System.

[Dur87, Dur88] discuss the method *partial global planning*, a flexible framework for coordination. It addresses the trade-off between predictability (related to quality) and responsiveness and the effect on coordination and system coherence. Coordination requires predictability. If unable to predict other's actions, agents cannot coordinate their interactions. Coordination is therefore easier when agents commit themselves to explicit, globally known plans. However, committing to such plans prevents agents from dynamically responding to

unexpected situations. To let a multi-agent system work effectively in dynamic domains, agents must be responsive, and thus unpredictable to a certain extent. Driving factor of partial global planning is the level of detail, from superficial but flexible with respect to unexpected events, to fully worked out but inflexible. The trend is to plan in detail for short-term actions, and superficial for long-term actions. Basically, this method addresses all aspects of real-time operation.

A strategy that will improve real-time performance by reducing multi-agent network communication is to incorporate *network awareness* in agents. Each agent simulates and predicts the activity of other agents through agent modelling. Communication can be reduced by e.g. focused addressing, duplication of processing, and monitoring agent behaviour. This reduction of communication is at the cost of increased local computation. Therefore, the system will function best if optimal trade-off is made between communication and computation.

[Les89, Kui94] discuss a promising technology area in DPS: *approximate reasoning*. The technology is often associated with blackboard systems. Approximate reasoning uses multiple methods for solving the problem. Each method (or approximation, which can either be a conventional or AIP method) makes a trade-off between the time required to generate the response and the quality of the response. Given a set of methods and their predicted execution lengths, the solution method which results in the best approximation (= highest quality of response) according to the time available is selected during run-time. [Dec93] calls this approach design-to-time scheduling. This enforces deadlines to be estimated accurately.

Processing speed-up can be achieved by reducing the solution quality along one of the following dimensions:

- *Completeness*: some solution aspects are ignored.
- *Precision*: some solution parameters are not determined exactly.
- *Certainty*: some supporting evidence is not considered.

[Les89] identifies three types of approximation approaches:

- *Approximate search strategies*, resulting in exploration of a *smaller* portion of the search space (e.g. [Mor92]).
- *Data approximations*, provide an abstract view of data resulting in a *simpler* space being searched (e.g. [Vin91]).
- *Knowledge approximations*, simplifying the inference process being applied in the system so that the search space can be explored more quickly.

Another method often associated with blackboard systems is *progressive reasoning*. The idea behind progressive reasoning is to produce a coarse-grained solution as fast as possible and then

refine it incrementally until all available time has been spent [Mic86]. For example [Lat86], a knowledge source may be divided into parts, where every part goes into more detail about the problem. When a knowledge source part has been evaluated and there is still time left, another deeper knowledge source part is evaluated in order to try to produce a better result. Note that with approximate processing the knowledge source that can give the best solution within the deadline is chosen *beforehand*, and with progressive processing the shallowest knowledge source is used first, followed by the processing of as many deeper levels as possible until the deadline is reached. No prediction on execution times is needed. However, problems are [Kui94]:

- Previous computations can not be used entirely, resulting in inefficiency.
- Problem decomposition in progressive reasoning levels might not be easy or even impossible.

Progressive processing techniques belong to the class of *any-time algorithms*. Any-time algorithms [Dea88] comprise a class of approaches that guarantee a response within any time. The computations are expected to return better responses when given more time. Note that the term "any-time algorithm" denotes a class of approaches and is not a particular prescription. Any-time algorithms are a flexible computation means since these algorithms can be interrupted at any point and always supply a response. However, two problems with any-time algorithms (and therefore also with progressive reasoning) appear:

- The discontinuity of expected response quality as a function of time (it is a "step" function).
- No provisions for coping with increasing event rate or number of operations (how long will the "step" take).

These problems could make any-time algorithms producing responses of a very poor quality under specific conditions.

Multithread reasoning integrates both approximate and progressive processing [Kui94]. This is done by implementing more than one way to reach an answer (threads varying in detail) and evaluate them independently in *parallel*. Short threads will provide global answers in short time whereas long threads will provide answers in detail consequently using more time. All threads start at the same time, and when no time is left, the best quality thread that has finished is chosen to provide an answer. Multi-thread reasoning could be implemented in (fine-grained) multi-agent systems. The multithread reasoning technique seems to be promising because it takes the advantage of approximate as well as progressive processing. It uses multiple methods to reach the best solution possible without efficiency loss (advantage of approximate processing). Additionally, it works as an any-time algorithm where no prediction on execution times is needed (advantage of progressive processing).

[Hay94] addresses the basic question how an agent can execute - with limited resources - high quality operations in bounded time, despite increases in event rate and number of known

operations. An extended blackboard architecture is designed with a *satisfycing cycle algorithm* for its reasoning cycle in real-time. The presented approach compromises between the performance aspects criticality, number of events to respond to (highly critical first), the quality and the latency of these responses in order to maintain a global utility of the system's behaviour over time.

[Ing93] remarks that the real-time extensions to the blackboard as presented in [Hay94] has its problem with respect to modularity (unclear what module does what function), event processing when buffers are full (forgets events), and large schedular overhead. [Ing93] argues that the implementation of the blackboard in REAKT [Lal92] gives better performance because of interruptable knowledge sources, the "intention" concept and the RETE algorithm (a typical example of a compilation algorithm).

4.2.3 Conclusion

Much research is being performed on real-time aspects in DPS at the level of both multi-agent systems and blackboard systems. Several approaches have been developed or are under research. Most approaches address the basic trade-off between criticality, quality and responsiveness, given available resources and deadlines. Previous sections have described some of the promising methods and techniques and indicated strong and weak points.

None of the discussed approaches are excluded from implementation in a real-time system on beforehand. However, the tendency is (see also section 3.4.3) to let a multi-agent system form the backbone architecture of a system that considers the basic trade-off between communication and computation and the asynchronity of coordination (in the context of deadlines, e.g. through PAO*), and to apply blackboard system technology to local problem solving (within an agent, perhaps serving as a backbone structure of an agent) that addresses the problem-dependent trade-off between quality and responsiveness.

4.3 Modularity

As remarked in section 4.1, multi-agent systems have a high degree of modularity. The actual processing is encapsulated in the agent itself, i.e. the internal processing and data structures of the agent are hidden from the other agents.

The ability to structure a complex problem or task into relatively self-contained processing modules (agents) leads to a modular system. Each agent may be specialized in solving a particular aspect of the problem. Through cooperation among these specialized agents, a solution for the overall problem is found. The inherent modularity of DPS allows a system to be constructed in a parallel, incremental and evolutionary way (in both the engineering phase and maintenance phase). It allows for scalability, extensibility, maintainability, and adaptability due

to reduced complexity of agents performing a relatively simple sub-task, and because knowledge and related processing is localized in a single expert or agent.

4.4 Integrability

Distributed problem solving allows for a highly modular, heterogeneous approach and is perfectly suited to integrate AI techniques with conventional programming techniques. Each agent might have its own knowledge representation, reasoning capabilities, data bases, etc. It allows for integration of all kinds of methods and techniques, because these will be encapsulated in an agent and hidden from the external agents. Blackboard systems are less heterogeneous, but still allow for different knowledge and data representation techniques. Below, a number of integrable technologies are discussed.

Knowledge-based systems (KBS) are often an integral part of DPS. [Lan94, Les81] argues that real-time KBSs include concepts that can be the basis for real-time distributed problem solving. Example integrable KBS methods beside knowledge representation and reasoning (inference and control) are temporal reasoning, non-monotonic reasoning, truth (or believe) maintenance [Gal91, Mas89], uncertainty reasoning [Par93], abstraction, search techniques, focus of attention strategies, and knowledge compilation (e.g. RETE).

The *object-oriented* (OO) approach matches well with DPS technology. Agents can be considered as very large objects. [Sch90] discusses integration of OO at the level of distributed (network) systems. [Hyn89] presents a developed framework that integrated OO with DPS by means of frame systems. [Har92] does this too (see section 4.5).

DPS has also been associated with *learning techniques*, although they are being barely discussed in literature. [Sha89] discusses adaptive learning in multi-agent system through the use of genetic algorithms. The approach aims at improving agents' knowledge and skill and the performance of the whole multi-agent system. It is based on the market mechanism in which agents compete through a bidding process, and a genetic transformation scheme which makes the system adaptive. The competitive learning process would help the multi-agent system to adapt to its environment.
In a multi-agent system, two types of learning may occur: the agents can learn as a group, while at the same time, each agent can also learn on its own by adjusting views and actions. At the group level, learning takes effect in the form of better coordination (e.g. information sharing, knowledge sharing and efficient signalling among agents), and more efficient task decomposition and distribution (e.g. by learning specialisations of agents, group characteristics, task patterns, and environmental characteristics such as user preference and machine reliability). At the individual agent level, learning activities could be related to improvement of own problem-

solving skills and/or to observation of problem-solving of other agents.
Be aware that incorporating learning capabilities in a system might increase non-determinism and unpredictability, which might be highly undesirable as discussed in section 4.1.

Planning techniques for incorporation in DPS is often addressed in literature. Some techniques have been discussed in section 4.2.2.

In conclusion, DPS provides rich concepts for easy integration of all kinds of methods and techniques, conventional as well as advanced information processing. Blackboard systems and multi-agent systems are often advertised as heterogeneous systems that integrate different kinds of software approaches. With respect to hardware integration, multi-agent systems are also easily integrable with - possibly distributed - multi-processor networks (in fact, this is often a requirement or necessity with respect to performance).

4.5 Engineering methodology

Section 4.5.1 discusses general aspects of system engineering with respect to the application of DPS technology. Section 4.5.2 focuses on a specific important aspect of engineering of DPS applications: the development of a user interface, being a critical aspect in user acceptance.

4.5.1 DPS system engineering

The application of DPS, and in particular multi-agent systems, has a number of potential advantages with respect to system engineering:

- Enhanced modularity.
- Concurrent/parallel development.
- Incremental/evolutive development (e.g. prototyping, system evolution).
- Increased maintainability, scaleability, extendability, adaptability, etc.

The specific issue of adopting a design methodology for DPS systems is rarely addressed in literature. However, the application of DPS in complex systems does not require a complete other system engineering approach. The usual system engineering methods (e.g. waterfall model) can be used as basis, but should allow for prototyping of the system or parts of the system (i.e. agents). Prototyping is necessary in order to discern and manage complexity and provide feedback to potential users. Prototyping and evaluation should be possible at both system architectural level, addressing the cooperative aspect, and agent level. The latter can be done concurrently for each agent. Architectural prototyping can be done separately from agent prototyping if the agent's characteristics are known to some extent. The SAHARA tool [Bar90] is an example of a tool that is used to prototype and evaluate alternative architectures, evaluating real-time performance and behaviour based on parameters such as agent organization, granularity,

resource availability, etc. This tool has been used for the crew assistant application Copilote Electronique [Cha89].

In general, it is strongly recommended to support the development of a complex DPS system with an extensive DPS/multi-agent toolkit (see section 4.6). Such a toolkit should hide much of the complexity of inter-agent communication, and agent's representation and processing from the developer. This will enable him to concentrate on the important, application domain-dependent aspects of distributed problem solving. The usage of these tools should reduce design complexity and allow for prototyping.

The tools should not only support a computational model to build DPS systems, but should also pursue a clear design methodology that considers prototyping, production and maintenance. [Har92] writes that an agent-oriented design methodology should consider the following criteria in order to built a modular system:

- *Agent decomposability*, the degree to which a design method assists in the decomposition of agents into subordinate agents.
- *Agent composability*, the degree to which a design method support the composition of pre-existing agents, groups, teams, etc., to be combined into various organisations (reusability).
- *Agent understandability*, the degree that an agent is understood by only considering that agent.
- *Agent continuity*, the design method should produce agents such that small agents do not require modification of other agents.
- *Agent protection*, the design method should support agent protection such that an agent failure does not imply failure of other agents.

[Har92] combines these agent-oriented design principles with object-oriented design principles and obtains the following set of criteria for a DPS system design methodology:

- Agent-oriented design principles as listed above,
- Persistence.
- Information hiding.
- Abstraction.
- Inheritance.
- Polymorphism.

Several tools already incorporate these aspects (such as MADE [Har92]).

In conclusion, development of DPS applications should be supported by a toolkit that has a clear design methodology preferably based on aspects as discussed above, and should allow for prototyping, incremental development, evaluation (including verification, debugging), and easy

maintenance.
Although life cycle aspects are rarely discussed in DPS-related literature, it is commonly understood that especially multi-agent systems have a positive contribution to life cycle costs due to the high degree of modularity.
Finally, because DPS is closely related to knowledge-based systems, the engineering methodology should also consider knowledge-based system engineering approaches. So, DPS system engineering should be a migration of conventional, object-oriented and knowledge-based system engineering methodologies with additional agent-specific features.

4.5.2 User interface engineering

In a DPS system, and in particular a multi-agent system, *user interfacing* emerges as an important aspect. As [Hol94] argues, the underlying aim of research in DPS can be seen to be the enhancement of the cooperation and coordination between distributed autonomous agents and an effective integration of man and machine. This integration should involve machines performing the tasks that they are best at, complemented by humans doing the jobs at which they excel. Only in such systems we can find a true optimization of all components.
[Avo92] provides a clear overview of the issues of user interface design for DPS applications. Systems are classified according to their user interaction characteristics in five groups:

- *Geographically distributed systems*: user interaction occurs at the individual node level where the user has access to the agent's local reasoning and its view of the rest of the network that affects its reasoning.
- *Reactive systems and simulators*: fine-grained graphic representation of the environment where objects - reactive agents - depict themselves with associated attribute information.
- *Functionally decomposed systems*: coarse-grained, complex, heterogeneous agents that interact through a dedicate interface control agent that schedules the dialogue, and hides the distribution aspect from the user by providing functional views.
- *Cooperative environments*: maintain interaction with groups of people.
- *Experimental testbeds*: user (developer) interaction has as purpose to study the behaviour of the system and agents.

Characteristics affecting user interaction are:

- *Agent granularity*: user interaction with a fine-grained system happens at overall system level, whereas with coarse-grained systems, user interaction is more towards individual agents in order to have the possibility for effective participation of the user in the problem solving.
- *Control*: the user can either be a member of the organization or be a supervisor, a static hierarchical control organization is easier to map into the conceptual model of the user, than a dynamic control organization which makes interface design complex.
- *Cooperation strategies*: in the task-sharing approach where the mapping of tasks to agents

is clear, the user will have a better understanding, than in the result-sharing approach with partial solutions that are not easy to understand and still have to be integrated.
- *Knowledge heterogeneity*: although agents might be heterogeneous, this should be hidden from the user, and the agents should interact with the user in a uniform way with common semantics.
- *Explanation*: at both agent and system (cooperative, group behaviour) level, explanation information should be provided. Due to expected distinction between agents and complex coordination, a dedicated node that builds distributed explanation information may be needed.

[Avo92] presents a user interface agent architecture as it is implemented in ARCHON [Jen92] that is based on these user interaction classification and characteristics. It has two main functions:
- Representation of agents and their problem solving to the user.
- Representation and modelling of the user within the system.

[Hal92] presents a methodology for user interface design for existing multi-agent systems.

It is advised to have one agent being responsible for user interaction. Main threads for user interface design are operator modelling (information needs and actions), and hiding of the complexity of the multi-agent system. Various functional views of the system should be available to the user, which map preferably on a particular agent (and its believes). If information from other agents is to be supported by the functional view, than this particular agent should provide that information from its point of view. This enables an integrated, agent-focused view of an agent's local reasoning and global system behaviour.

Another important concept to be considered is that natural objects (part of the system's environment) graphically depict themselves and behave as reactive agents on which the crew can perform operations. Representation of agents and the corresponding processes as spatially organized objects (e.g. engine, threat) that have size and shape, can serve as interlocutor cue in a mental environment, so that the user understanding and interaction with the system is facilitated by the use of these spatial landmarks within a realistic representation of the problem solving world. Interaction based on realistic images will speed up (or relax the boundaries of) the user's cognitive processing (refer to the limited resources, section 3.2.2).

4.6 Maturity and next generation

Maturity can be measured by the availability of tools and the realisation of operational applications. Section 4.6.1 provides insight in the former, and section 4.6.2 in the latter, in particular in relation with the aerospace domain. Section 4.6.3 provides a statement on next generation.

4.6.1 Tools

A rich set of tools are currently on the market, so that in this respect maturity and state-of-the-art of DPS is relatively high. The blackboard concept is older than the multi-agent concept. But because of the nicer properties of multi-agent systems, companies are focusing more and more on multi-agent technology which is reflected by the growing list of multi-agent development tools. In fact, most of these tools have integrated blackboard system technology in their architecture.

On the other side, much research is still to be done. To provide another - less positive - indication of maturity, [Gas89] mentions a number of research themes and associated problems that have to be solved or worked out:

- Methodology (DPS engineering is still an art).
- Deep theories of coordination (yet too specialized and project-specific).
- Representation of collective actions (complex coordinated actions).
- Learning (e.g. adaptive organization self-design).
- Multi-grain concurrency (e.g. integration neural networks and high-level distributed symbolic reasoning).
- Modelling and explaining problem-solving behaviour (explanation, prediction, analysis).
- Hypothetical worlds (different views, conflict resolution).
- Real-time artificial intelligence (computation, communication, resource management).
- Epistemology and emergent knowledge (e.g. knowledge sharing).

Research and the development of tools should address these themes to allow for a next generation DPS applications. This section will present a number of evaluation criteria for evaluation and selection of DPS tools for a complex real-time application. Furthermore, a list of candidate tools is provided, divided in blackboard tools and multi-agent tools. Actual evaluation and selection can be performed if the application and its specific requirements are known.

Evaluation criteria.

The following criteria are of importance for DPS tool selection:

- Supports a system engineering methodology (prototyping, concurrent development).
- Object-oriented design principles.
- Support for user interaction (user modelling, graphical distributed user interfaces).
- Minimizes life cycle costs (easy maintenance, extendability).
- Integration multi-processor technology.
- Representation of problems/tasks.
- Decomposition of problems/tasks.
- Prioritisation of problems/tasks.

- Agent planning.
- Task allocation.
- Resource allocation and management.
- Definition of organization roles.
- Communication management.
- Data abstraction.
- Meta-level information.
- Performance evaluation based on metrics (e.g. see section 3.3.3).
- Testability (debugging tools).
- Handling of asynchronous and unpredictable events.
- Handling of dynamically changing data during problem solving (non-monotonicity, focus of control).
- Time constraints and the trade-off with the response quality (time-constrained reasoning).
- Reasoning about events in both space and time (time-stamped data: creation-time, validity time; temporal model of the system, temporal and spatial reasoning).
- Handling of uncertainty and sensor data (uncertainty reasoning, believe revision).
- Continuous operation (e.g. history management, garbage collection).
- Handling of deadlines (guaranteed response time, hard real-time requirements and any-time requirements).
- Agent inhibition mechanism.
- Agent/architecture compilation.

Candidate blackboard tools.

- ATOME [Laa89].
- Erasmus (BB1) [Kai89].
- GBB [Cor89b].
- GEST [Bau89].
- RT-1 [Dod89].

Candidate multi-agent tools.

- ABE [Hay88] (applied in Pilot's Associate).
- AF [Gre87].
- Agora [Bis87].
- ARCHON [Jen92].
- CADDIE [Ma].
- CAGE [Nii89].
- CIRCA [Mus93].
- GBB (extended) [Cor89b].
- KOS [Heu91] (applied in Copilote Electronique).

- MACE [Gas87].
- MADE [Har92].
- MAGES [Bou91].
- MUSE [Rey87].
- POLIGON [Nii89].
- RT-SOS [Mou93].
- SOCIAL [Adl89b].
- SPLICE [Mul93].

4.6.2 Applications

The late 80s and these early 90s have brought a number of applications using DPS technology. Some of these are even operational. For example, [Coh89] discusses the forest fires control system Phoenix. It applies real-time, adaptive planning with approximate scheduling algorithms and distributed planning techniques. [Har90] discusses the distributed system STORMCAST for forecasting severe storms over Scandinavia. Also in the area of avionics, DPS technology emerges. The remainder of this section provides a list of applications from the aerospace domain, focused on crew assistant type of application. Extensive literature is available on applications in other domains, e.g. robotics, computer networks, multi-sensor data fusion, command and control.

[Bau89] discusses the *Cockpit Information Management* prototype system CIM that uses a blackboard architecture as basis. The blackboard approach is chosen because it supports responses that are context sensitive, partitioning of problem-solving process, hierarchical, abstract organization in data and domain knowledge, and separation of data, knowledge, functionality and control. A basic blackboard model has been upgraded in order to meet efficiency (real-time) requirements. The enhancements consider focus of attention, efficient blackboard representation, asynchronous input, interrupting the control cycle, and exploiting parallelism, and in the long-term: guaranteed response time, reasoning with time, uncertainty reasoning, and improved performance through compilation. It employs a combination of the Erasmus blackboard (with the BB1 control regime) and GGB that proved to be much more efficient with respect to real-time performance (factor 5) than Erasmus alone. [Ble89] provides a benchmarking report of blackboard systems for this application.

The *Pilot's Associate* makes use of blackboard system as well as multi-agent technology. [Ban91] writes that the software of Pilot's Associate has been structured in a heterogeneous, loosely coupled system in which individual systems are not restricted to a particular development environment or software approach. [Cor89] reports that Pilot's Associate adopts a distributed blackboard architecture.
Communication between modules is centrally coordinated by a sub-system called the Mission

Manager. It maintains a global blackboard which is the central repository for active plans and goals (i.e. centralized multi-agent planning). Basic tool that is used is the ABE/RT real-time tool [Hay88] (see also [Hin94]).
The system status function of the Pilot's Associate is also organized as a blackboard system, adopting the blackboard control architecture as described in [Hay85]. [Pom90] provides performance results and problems of this application. Note that [Ban91] states that the centralized approach has been abandoned for complexity and performance reasons and is being decentralized and distributed among the agents.

[Ger87] discusses a prototype application of an *expert system that manages a set of cooperating expert systems*. It provides interaction management towards the multiple expert systems as well as interaction management towards the pilot, so that the complexity of the multi-expert system is hidden from the pilot.

[Cha93] writes that the technical specification of the *Copilote Electronique* required a flexible heterogeneous implementation paradigm. In this respect, a multi-agent system architecture using DPS techniques has been chosen as basis. Basic design parameter considered with this project is the way cognitive agents interact. Multiple alternatives are possible which requires a careful evaluation at an early stage of the structure of the architecture in three major aspects:

- *Functionality*, where the system is verified that it will achieve the desired behaviour and uses the proper information.
- *Structure*, where the system organisation is analyzed on aspects such as decomposition and clustering (e.g. agent granularity), and communication flows (e.g. shared data, message passing, etc.) to master communication flows and process control.
- *Virtual resources*, where system performance is evaluated with respect to processor power, multiple processors, memory capacity and I/O channels.

This evaluation is performed with the simulation tool SAHARA [Bar90]. The basic tool on which Copilote Electronique, and therefore the multi-agent system architecture, will be implemented is the Knowledge-based Operating System (KOS) tool [Gil92].

Furthermore, [Rob87] reports a *Mission Management Aid* application TACAID that has been developed with the Muse blackboard system tool [Rey87]. An object-oriented approach has been adopted. The *Threat Management System* as reported in [Hol91] uses also blackboard system concepts.

4.6.3 Next generation

The potency of DPS technology, blackboard systems as well as multi-agent systems, has already been recognized in the research community and industry (including aerospace). Much research is spent on DPS, and in particular on real-time multi-agent systems. A rich set of tools is already available which will be improved in the future and new advanced tools will appear on the market. Because of the nice features of DPS and the progress in research, tools, applications and multi-processor technology that is currently being made, it can be concluded that DPS systems based on multi-processor technology will play a dominant role in next generation advanced information processing technologies and complex modular critical applications.

5 Conclusion

DPS has a number of features to manage the complexity of real-time critical systems as employed in the aerospace domain. Therefore, it is recommended to consider application of DPS technology in complex modular critical applications for the following reasons:

- Modularity, reduced complexity and reduction life cycle cost.
- Concurrent and incremental development.
- Inherent distribution of the application (functional).
- Integration of heterogeneous systems.
- Reliability.
- Easy mapping of task domains on agents.
- Considers the limited availability of resources.
- Data abstraction.
- Handling of bounded response times and reasoning.
- Real-time behavioral characteristics.

Below, a summary of the evaluation by means of a number of criteria relevant to a complex modular critical application is provided.

Functionality. From a functional point of view, relating DPS to complex modular real-time systems shows that blackboard systems and multi-agent systems are relatively made-to-measure technologies. These technologies can be applied to both component and system level.

Reliability. DPS technology will increase reliability (and safety) of systems if non-determinism is kept to an absolute minimum. Total safety is only guaranteed if the following conditions are satisfied:

- The system's task is to *support* the user.
- The user will always be in command as final *authority*.
- Delegated autonomous operation may only be considered for simple, routinely tasks that ensure deterministic and predictable agent behaviour.

Performance. Much research is being performed on real-time aspects in DPS, both at the level of multi-agent systems and blackboard systems. Most approaches address the basic trade-off between criticality, quality and responsiveness, given available resources and deadlines. The tendency is to let a multi-agent system form the backbone architecture of an application that considers the basic trade-off between communication and computation and the asynchronity of coordination, and to apply blackboard system technology to local problem solving (within an agent) that addresses the problem-dependent trade-off between quality and responsiveness.

Modularity. Decomposition of an application by task domain, level of processing and/or problem domain will form the basis for a modular system architecture of multiple cooperating agents. It allows for development and maintenance in a structured manner in order to be able to anticipate to the ever changing operational environment and user demands.

Integrability. DPS provides rich concepts for easy integration all kinds of methods and techniques, conventional as well as advanced information processing. Blackboard systems and multi-agent systems are often advertised as heterogeneous systems that integrate different kinds of software approaches. Because of this heterogeneity, DPS system engineering should be a migration of conventional, object-oriented and knowledge-based system methods and techniques.

System Engineering. Although a reduction of complexity by having a system decomposed in multiple cooperative agents is promised, the overall complexity of applying DPS to should not be underestimated. The way of coordination and achieving coherence remains complex and needs deliberate system design. In order to control complexity as well as to achieve the required performance, decomposition, distribution and cooperation strategies should not be too flexible. In fact, it is argued to embed fixed strategies in the design of the system by the designer himself, rather than letting the system itself dynamically apply strategies introducing overhead and possibly incoherent (non-convergent or non-predictable) behaviour.

To avoid this harmful system behaviour, the following measures can be taken for system development:

- Develop the system incrementally (range of prototypes) to increase functionality and performance step by step.
- Provide a good development environment (an advanced DPS toolkit is required).
- Apply decomposition on basis of a formally prescribed task or problem hierarchy.
- A priori known distribution of tasks among agents.
- Avoid conflicts between agents.
- Design a fixed community-like organization of agents with strict rules of behaviour based on identified task/problem domains.
- Reduce non-determinism.
- Make use of next generation on-board hardware resources based on multi-processor technology.
- Apply resource management (see section 3.2.2).

With respect to an engineering methodology, the heterogeneous aspect of DPS technology, system engineering should be a migration of conventional, object-oriented and knowledge-based system engineering methodologies with additional agent-specific and user interaction design

features. It should allow for incremental development (prototyping).

Maturity and next generation. The potency of DPS technology, blackboard systems as well as multi-agent systems, is recognized in the research community and industry (including aerospace). Much research is spent on DPS, and in particular on real-time multi-agent systems. A rich set of tools is already available which will be improved in the future and new advanced tools will appear on the market. Prototypes and operational applications have already been built with DPS technology.
Because of the nice features of DPS and the progress in research, tools, applications and multi-processor technology that is currently being made, it can be concluded that DPS systems based on multi-processor technology will play a dominant role in next generation advanced information processing technologies.

Considering the evaluation of the criteria, it is recommended to consider DPS technology in complex modular (decomposable) critical systems and let it be a driving technology for the overall system architecture.

6 References

[Adl89a] Adler M.R., et al., *Conflict Resolution Strategies for Non-Hierarchical Distributed Agents*, in *Distributed Artificial Intelligence Vol. II*, Gasser L., Huhns M.N. (eds.), London Pitman/Morgan Kaufmann, 1989.

[Adl89b] Adler R.M., Cottman B.H., *A Development Framework for Distributed Artificial Intelligence*, Proc. of the 5th Conf. on Artificial Intelligence Applications, Miami, 6-10 March, 1989.

[And85] Anderson B.M., et al., *Expert Systems for Aiding Combat Pilots*, S-9185-0916D, 1985.

[Avo92] Avouris N.M., *User Interface Design for DAI Applications: An Overview*, Distributed Artificial Intelligence, Theory and Praxis, Eds. Gasser, Avouris (Eds.), Boston Kluwer Academic, 1992.

[Ban91] Banks, S.B., Lizza C.S., *Pilot's Associate: A cooperative Knowledge-Based System application*, IEEE Expert, June, 1991.

[Bar90] Barat et al., *Design and Analysis of Multi-Agent Systems with Sahara Simulation Tool*, COGNITIVA 90, 1990.

[Bau87] Baum L.S., et al., *Advanced Blackboard Approaches for Cockpit Information Design*, S-1987-1005D, Boeing Computer Services, p. 15-30, 1987.

[Bau89] Baum L.S., *Recent Developments in Blackboard Frameworks (Foreword)*, Blackboard Architectures and Applications, p. 303-308, 1989.

[Bis87] Bisiani R., et al., *The Architecture of the Agora Environment*, Distributed Artificial Intelligence (Ed. Huhns), p. 99-117, 1987.

[Ble89] Blevins D.M., et al., *Benchmarking Blackboards to Support Cockpit Information Management*, 89-3095-CP, Boeing Computer Services, 1989.

[Bon88a] Bond A.H., Gasser L. (eds.), *Readings in Distributed Artificial Intelligence*, San Mateo, CA, Morgan Kaufmann, 1988

[Bon88b] Bond A.H., Gasser L., *An Analysis of Problems and Research in DAI*, Bond A.H., Gasser L. (eds.), *Readings in Distributed Artificial Intelligence*, San Mateo, CA, Morgan Kaufmann, 1988

[Bon92] Bond A.H., Gasser L., *A Subject-Indexed Bibliography of Distributed Artificial Intelligence*, IEEE Transactions on System, Man and Cybernetics, Vol. 22, No. 6, Nov 1992.

[Bou91] Bouron T., et al., *MAGES: A Multi-Agent Testbed for Heterogeneous Agents*, Decentralized AI - 2, Demazeau, Müller, p. 195-214, 1991.

[Cha93] Champigneux G., *Flight Mission Planning in the Co-pilote Electronique*, AGARD Lecture Series on New Advances in Mission Planning and Rehearsal Systems, Oct. 1993.

[Coh89] Cohen P.R., et al., *Trial by Fire: Understanding the design Requirements for Agents in Complex Enviroenments*, AI Magazine, p. 33-48, 1989.

[Cor89a] Corrigan, J. and K. Keller, *Pilot's Associate: An inflight mission planning application*, Proceedings of the AIAA Guidance, Navigation and Control Conference, Boston, MA, 1989.

[Cor89b] Corkill D.D., *Design Alternatives for Parallel and Distributed Blackboard Systems*, Blackboard Architectures and Applications, p. 99-136, 1989.

[Dav83] Davis R., Smith R.G., *Negotiation as a Metaphor for Distributed Problem Solving*, Artificial Intelligence, Vol. 20, No. 1, p. 63-109, 1983.

[Dec87] Decker K.S., *Distributed Problem Solving Techniques: A Survey*, IEEE Transactions on Systems, Man, and Cybernetics, Vol. 17, No. 5, p. 729-40, 1987.

[Dec89] Decker K.S., et al., *Evaluating Research in Cooperative Distributed Problem Solving*, in *Distributed Artificial Intelligence Vol. II*, Gasser L., Huhns M.N. (eds.), London Pitman/Morgan Kaufmann, 1989.

[Dec93] Decker K.S., et al., *A Real-Time Control Architecture for an Approximate Processing Blackboard System*, Int. Journal of Pattern Recognition and Artificial Intelligence, Vol. 7, No. 2, p. 265-284, 1993.

[Dem90] Demazeau Y., Muller J.P., *Decentralized Artificial Intelligence*, New York, Elsevier, 1990.

[Dem91] Demazeau Y., Muller J.P., *From Reactive to Intentional Agents*, Decentralized AI - 2, Demazeau, Müller, p. 3-10, 1991.

[Die90] Dieng R., *Temporal Relations linking Cooperating Expert Systems*, Proc. 5th Int. Symp. Methodologies for Intelligent Systems, p. 51-58, Nashville, TN, Oct., 1990.

[Die91] Dieng R., *Relation linking Cooperative Agents*, Decentralized AI - 2 Eds. Demazeau, Müller, p. 51-57, 1991.

[Dea88] Dean T., Boddy M., *An Analysis of Time Dependent Planning*, in: Proceedings AAAI, p. 49-54, 1988.

[Dod89] Dodhoawala R.T., et al., *A Real-Time Blackboard Architecture*, Blackboard Architectures and Applications, p. 219-237, 1989.

[Dur87] Durfee E.H., Lesser V.R., *Using Partial Global Planning to Coordinate Distributed Problem Solvers*, Proc. of 1987 Int. Joint Conf. on Artificial Intelligence, p. 875-888, 1987.

[Dur88] Durfee E.H., Lesser V.R., *Predictability versus Responsiveness, Coordinating Problems Solvers in Dynamic Domains*, Prc. of the 7th Nat. Conf. on Artificial Intelligence, 1988.

[Dur89] Durfee E.H., et al., *Trends in Cooperative Distributed Problem Solving*, IEEE Transactions on Knowledge and Data Engineering, Vol. 1, No. 1, March, 1989.

[Eri85] Erickson W.K., *The Blackboard Model: a Framework for Integrating Multiple Cooperating Expert Systems*, NASA/Ames Research Center, Moffet Field, California, 1985.

[Erm80] Erman L.D., et al., *The Hearsay-II Speech Understanding System: Integrating Knowledge to Resolve Uncertainty*, Comput. Surveys, Vol. 12, No. 2, p. 213-53, June, 1980.

[Faw91] McFawn L, Morgan D.R., *Avionics Technology Beyond 2000*, AGARD Lect. Series 176, The Conflicting Forces Driving Future Avionics Acquisition, 1991.

[Gal90] Galliers J.R., *Modelling Autonomous Belief Revision in Dialogue*, In: Decentralized AI - 2, Demanzeau Y., Müller J.P. (Eds.), 1991.

[Gas87] Gasser L., et al., *MACE: A Flexible Testbed for Distributed Artificial Intelligence*, Distributed Artificial Intelligence (Ed. Huhns), p. 119-152, 1987.

[Gas89] Gasser L., Huhns M.N., *Distributed Artificial Intelligence Vol. II*, London Pitman/Morgan Kaufmann, 1989.

[Gas92a] Gasser L., *An Overview of DAI*, Distributed Artificial Intelligence, Theory and Praxis, Eds. Gasser, Avouris (Eds.), Boston Kluwer Academic, 1992.

[Gas92b] Gasser L., *DAI Approaches to Coordination*, Distributed Artificial Intelligence, Theory and Praxis, p. 31-51, 1992.

[Ger87] Gerstenfield A., et al., *An Expert System for Managing Cooperating Expert Systems*, Artificial Intelligence in Engineering: Tools and Techniques, Cambridge, MA, Aug., 1987.

[Gil92] A. Gilles, et al, *A Parallel Multi-Expert Architecture for the Copilote Electronique*, Dassault-Aviation, France, 1992 (?).

[Goo83] Goodson J., et al., *Distributed Intelligence Systems: AI Approaches to Cooperate Man-Machine Problem Solving in C3I*, AIAA, 83-2316, 1983.

[Gre87] Green P.E., *AF: A Framework for Real-Time Distributed Cooperative Problem Solving*, Distributed Artificial Intelligence (Ed. Huhns), p. 153-175, 1987.

[Hal92] Hall L.E., Avouris N.M., *Methodological Issues of DAI Applications Interface Design: Transparancy Analysis*, Distributed Artificial Intelligence: Theory and Praxis, p. 163-78, 1992.

[Har90] Hartvigsen G, Johansen D., *Co-operation in a Distributed Intelligent Environment - The stormCAST Application*, Eng. Appl. of AI, Vol. 3, p. 229-237, 1990.

[Hay85] Hayes-Roth B., *A Blackboard Architecture for Control*, Artificial Intelligence, p. 251-321, July, 1985.

[Hay88] Hayes-Roth F., *ABE: A Cooperative Operating System and Development Environment*, Readings in Distributed Artificial Intelligence (Ed. Bond, Gasser), p. 457-489, 1988.

[Hay94] Hayes-Roth B., *A Satisficing Cycle for Real-Time Reasoning in Intelligent Agents*, Expert Systems with Applications, Vol. 7, p. 31-42, 1994.

[Heu91] Heudin J.C., *KOS: A Knowledge-Based Real-Time Executive for Embedded On-Board Applications of Artificial Intelligence*, ESA/ESTEC Workshop on Artificial Intelligence and Knowledge-Based System for Space, Noordwijk, The Netherlands, May, 1991.

[Hol91] Holla K., Benninghofen B., *A Threat Management System*, Proceedings of the AGARD Avionics Panel Symposium on Machine Intelligence for Aerospace Electronic Systems, Lisbon, Portugal, (AGARD-CP-499), 1991.

[Hol94] Holt J., Rodd M.G., *An Architecture for Real-Time Distributed Artificial Intelligent Systems*, Real-Time Systems, Vol. 6, p. 263-88, 1994.

[Hyn89] Hynynen J., Lassila O., *On the Use of Object-Oriented Paradigm in a Distributed Problem Solver*, AICOM Vol. 2, No. 3/4, p. 142-151, December, 1989.

[Ing93] Ingrand F.F., Coutance V., *Procedural Reasoning versus Blackboard Architecture for Real-Time Reasoning*, Proc. 13th Int. Conf. on Artificial Intelligence, Expert Systems and Natural Language, p. 449-58, Avignon, May 1993.

[Jag89] Jagannathan V., et al., *Blackboard Architectures and Applications*, Perspectives in Artificial Intelligence Vol. 3, Academic Press, Inc, 1989.

[Jen92] Jennings N.R., Wittig T., *ARCHON: Theory and Pactice*, Distributed Artificial Intelligence: Theory and Praxis, Eds. Gasser, Avouris, p. 179-95, 1992.

[Kai89] Kaiser K., et al., *Adapting the Blackboard Model for Cockpit Information Management*, Blackboard Architectures and Applications, p. 481-500, 1989.

[Kri87] Krishna C.M., et al., *Processor Tradeoffs in Distributed Real-Time Systems*, IEEE Transactions on Computers, Vol. C-36, No. 9, p. 1030-40, December, 1987.

[Kui94] Kuiper H., Van de Poll E., *Real-time Aspects of Advanced Information Processing in Multi-Sensor Data Fusion*, NLR Contract Report 94608 L, 1994.

[Laa89] Lâasri H., Maître B., *Flexibility and Efficiency in Blackboard Systems: Studies and Achievements in ATOME*, Blackboard Architectures and Applications, p. 309-322, 1989.

[Lal92] Lalanda P., et al., *A Real-Time Blackboard-Based Architecture*, Proc. of the 10th Eur. Conf. on Artificial Intelligence, 1992.

[Lan94] Lane D.M., McFadzean A.G., *Distributed Problem Solving and Real-Time Mechanisms in Robot Architectures*, Engineering Applications Artificial Intelligence, Vol. 7, No. 2, p. 105-117, 1994.

[Lat86] Lattimer Wright M., et al, *An expert for real-time control*, in: IEEE Software, p. 16-24, March, 1986.

[Lek92] Lekkas G., Van Liedekerke M., *Prototyping Multi-Agent Systems: A Case Study*, Distributed Artificial Intelligence, Theory and Praxis, Eds. Gasser, Avouris, Boston Kluwer Academic, p. 129-140, 1992.

[Les81] Lesser V.R., Corkill D.D., *Functionally Accurate, Cooperative Distributed Systems*, IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-11, p. 81-96, Jan., 1981.

[Les83] Lesser V.R., Corkill D.D., *The Distributed Vehicle Monitoring Testbed: A Tool for Investigating Distributed Problem Solving Networks*, AI Magazine, p. 15-33, Fall 1983.

[Les87] Lesser V.R., *Distributed Problem Solving*, in S.C. Shapiro, Ed., Encyclopdia of Artificial Intelligence, New York, Wiley, p. 245-251, 1987.

[Les89] Lesser V.R., et al., *Approximate Processing in Real-Time Problem Solving*, Blackboard Architectures and Applications, p. 239-268, 1989.

[Ma] Ma Z., et al., *CADDIE and its Multi-Agent Planner*, Logica Camebridge Ltd.

[Mar92] Von Martial F., *Coordinating Plans of Autonomous Agents*, Lecture Notes in Artificial Intelligence, Computer Science 610, 3-540-55615-x, Springer-Verlag, Berlin, 1992.

[Mas89] Mason C.L., Johnson R.R., *DATMS: A Framework for Distributed Assumption-Based Reasoning*, Distributed Artificial Intelligence, Vol. II (Eds. Gasser, Huhns), p. 319-383, 1989.

[Mic86] Michalski R.S., Winston P.H., *Variable Precision Logic*, in: Artificial Intelligence, Vol. 29, No. 2, p. 121-146, 1986.

[Mor92] Morgan K., Whitebread K.R., Kendus M., Crowmarty A.S., *Integration of Domain and Resource-Based Reasoning for Real-Time Control in Dynamic Environments*, in: The sixth annual workshop on space operations applications and research, p. 321-326, 1992.

[Mou93] Mouaddib, A.I., et al., *Real-time Message Engine for a Multi-Agent Architecture*, Proc. 13th Int. Conf. on Artificial Intelligence, Expert Systems and Natural Language, p. 589-99, Avignon, May 1993.

[Mul93] Mulder F.W., Boasson M., *A Loosely Coupled Distributed Blackboard System for Approach Control*, Proc. 13th Int. Conf. on Artificial Intelligence, Expert Systems and Natural Language, p. 459-71, Avignon, May 1993.

[Mus93] Musliner D.J., *CIRCA: A Cooperative Intelligent Real-Time Control Architecture*, IEEE Transactions on Systems, Man, and Cybernetics, Vol. 23, No. 6, Nov/Dec, 1993.

[Nii86] Nii H.P., *Blackboard Systems: the Blackboard Model of Problem Solving and the Evolution of Blackboard Architectures*, AI Magazine, Summer, 1986.

[Nii89] Nii H.P., et al., *Experiments on Cage and Poligon: Measuring the Performance of Parallel Blackboard Systems*, Distributed Artificial Intelligence, Vol. II (Eds. Gasser, Huhns), p. 319-383, 1989.

[Par93] Parsons S., Saffiotti A., *Integrating Uncertainty Handling Formalisms in Distributed Artificial Intelligence*, Proc. European Conf. on Symbolic and Quantitative Approaches to Reasoning and Uncertainty, p. 304-309, 1993.

[Pom90] Pomeroy B., Irving R., *A Blackboard Approach for Diagnosis in Pilot's Associate*, IEEE Expert, p. 39-46, Aug 1990.

[Pol93] Polat F., et al., *Distributed Conflict Resolution Among Cooperating Expert Systems*, Expert Systems, Vol. 10, No. 4, p. 227-36, Nov., 1993.

[Rau89] Raulefs P., *Toward a Blackboard for Real-Time Interactions with Dynamic Systems*, Blackboard Architectures and Applications, p. 285-299, 1989.

[Rey87] Reynolds D., *A Toolkit for Embedded, Real-Time AI*, Cambridge Consultants Ltd., March 17, 1987.

[Rob91] Roberts, K., *TACAID - A Knowledge-Based System for Tactical Decision Making*, AGARD AvP Symposium 61, Paper no. 7, May, 1991.

[Rod93] Rodd M.G., et al., *Architectures for Real-Time Intelligent Control Systems*, Information Infrastructure Systems for Manufacturing (Eds. Yoshikawa, Goossenaerts), p. 375-88, 1993.

[Ros92] Rosenschein J.S., *Teaching Distributed Artificial Intelligence*, Distributed Artificial Intelligence: Theory and Praxis, p. 215-227, 1992.

[Sch90] Schill A., *Distributed Application Support: Survey and Synthesis of Existing Approaches*, Information and Software Technology, p. 545-58, 1990.

[Sch94] Schwuttke U.M., et al., *Cooperating Expert Systems for the Next Generation of Real-Time Monitoring Applications*, Proc. 2nd Int. Conf. on Expert System for Development, p. 210-15, 1994.

[Sem91] Semple, W.G., *Development of Tactical Decision Aids*, AGARD AvP Sym 61, paper 17, May, 1991.

[Sha89] Shaw M.J., Whinston A.B., *Learning and Adaption in Distributed Artificial Intelligence Systems*, Distributed Artificial Intelligence, Vol. II (Eds. Gasser, Huhns), p. 319-383, 1989.

[Sik89] Sikka D.I., Varshney P.K., *A Distributed Artificial Intelligence Approach to Object Identification and Classification*, SPIE Vol. 1100 Sensor Fusion II, 1989.

[Smi81] Smith R.G., Davis R., *Frameworks for Cooperation in Distributed Problem Solving*, IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-11, p. 61-70, 1981.

[Smi88] Smith R.G., *The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver*, Reading in Distributed Artificial Intelligence, p. 357-366, Morgan Kaufmann Publishers Inc., San Mateo, California, 1988.

[Sri87] Sridharan N.S., *Report on the 1986 Workshop Distributed Artificial Intelligence*, AI Magazine, Vol. 8, No. 3, p. 75-85, Fall, 1987.

[Ste86a] Stenerson R.O., *Integrating AI into the Avionics Engineering Environment*, Computer, Feb., 1986.

[Ste86b] Steeb R., et al., *Cooperative Intelligence for Remotely Piloted Vehicle Fleet Control*, Tech. Rep. R-3408-ARPA, Rand Cooperation, Santa Monica, CA, Oct., 1986.

[Ste89] Stenerson R.O., et al., *Problem Focus Mechanisms for Cockpit Automation*, 89-3096-CP, Boeing Computer Services, 1989.

[Syc89] Sycara K.P., *Multi-Agent Compromise via Negotiation*, in *Distributed Artificial Intelligence Vol. II*, Gasser L., Huhns M.N. (eds.), London Pitman/Morgan Kaufmann, 1989.

[Ten81] Tenney R.R., Sandell Jr. N.R., *Strategies for Distributed Decision-Making*, IEEE Transactions on Systems, Man, and Cyberbetics, Vol. SMC-11, p. 527-38, 1981.

[Uma93] Uma G., et al., *Distributed Intelligent Systems; Issues, Perspectives and Approaches*, Knowledge-Based Systems, Vol. 6, No. 2, p. 77-86, June, 1993.

[Vin91] Vina A., Hayes-Roth B., *Knowledge-based real time control: the use of abstraction to satisfy deadlines*, in: IFAC Artificial Intelligence in Real-Time Control, California, USA, p. 33-40, 1991.

[Wes81] Wesson R., et al., *Network Structures for Distributed Situation Assessment*, IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-11, No. 1, Jan., 1981.

[Zui94] Zuidgeest R.G., *Multi-Sensor Data Fusion in Command and Control and the Use of Artificial Intelligence*, AGARD Symposium on Guidance and Control Techniques for Future Air-Defence Systems, 17-20 May, Copenhagen, 1994.