Nationaal Lucht- en Ruimtevaartlaboratorium
National Aerospace Laboratory NLR

NLR-TP-99308

Time-critical allocation of tactical air resources to targets

Y.A.J.R. van de Vijver

Nationaal Lucht- en Ruimtevaartlaboratorium
National Aerospace Laboratory NLR

NLR-TP-99308

Time-critical allocation of tactical air resources to targets

Y.A.J.R. van de Vijver

This investigation has been carried out under a contract awarded by the Ministry of Defence, contract number 9327.
The Ministry of Defence has granted NLR permission to publish this report.

This report is based on a presentation to be held at the NATO RTO Symposium "Advanced Mission Management and System Integration Technologies for Improved Tactical Operations", Florence, Italy, 27-29 September 1999.

Division:	Information and Communication Technology
Issued:	August 1999
Classification of title:	unclassified

Contents

(11 pages in total)

Time-critical Allocation of Tactical Air Resources to Targets

Yves van de Vijver
National Aerospace Laboratory NLR
P.O. Box 90502
1006 BM Amsterdam
The Netherlands
Tel: +31 20 511 3671
Fax: +31 20 511 3210
Email: vyver@nlr.nl

Abstract

In this paper, a tool for time-critical allocation of tactical air resources and artillery assets to a number of prioritised targets will be described. For air resources, both Close Air Support and Battlefield Air Interdiction missions are supported. Because of the general nature of the implemented solution, the tool could be tailored to other domains, for instance, the domain of unmanned aerial vehicles.

The tool, called RACAS, helps the planner to find feasible allocation plans. A first plan is provided instantaneously, while more and better plans are provided when time allows. This leaves the planner more time to compare the alternatives and select the plan most appropriate to the situation at hand. Furthermore, RACAS supports the planner in monitoring the situation. If an event occurs which will make the plan invalid, or degrade in quality, the planner will be notified. The tool will then support the planner in finding a plan for the new situation, if the planner wishes to do so.

RACAS has been developed with state-of-the-art techniques in software engineering, such as agent-based computing and object-oriented design. An implementation of RACAS has been demonstrated during exercise "Bright Sword" of the Royal Netherlands Army 1 (NL) Division "7 December" in June 1998.

1 Introduction

In this paper, the development of an agent-based, time-critical decision support tool, called the Resource Allocator for Close Air Support (RACAS), will be described. All parts of the adopted lifecycle, from analysis to implementation and evaluation, will be discussed, with an emphasis on design.

In designing RACAS, decomposition by means of separation of concerns has been applied to design agents that will be easy to reuse in other applications. The resulting design of RACAS consists of three co-operating agents. In design, agent based computing has been combined with object orientation to cover the various levels of detail. At the higher levels of abstraction, the approach is agent-based, at the lower level the approach is object-oriented.

RACAS has been implemented on top of a multi-agent architecture [2] developed within the EUCLID RTP 6.1 programme [8]. This programme, titled "Advanced Workstation for Command, Control, Communications, and Intelligence (C^3I)", started in September 1993 and finished in November 1998, and was carried out by a consortium of nineteen organisations from seven countries [5].

The objectives of EUCLID RTP 6.1 were the following:

- To demonstrate key artificial intelligence and human computer interaction methods and tools of particular relevance to advanced C^3I systems;
- To define and implement a workstation architecture able to accommodate existing and emerging methods and tools, and to forge new standards;
- To form domain simulations drawn from land tactical and naval systems to aid the development and testing of the architecture and its methods;
- To encourage European companies to work together effectively.

The paper has been organised as follows. In Section 2, the approach of the programme will be described in its historical context. In Section 3, the RACAS specification will be given. In Section 4, the system design of RACAS will be described. The RACAS application consists of three co-operating agents. The design of these agents will be described in Section 5. In Section 6, the construction of an ontology will be discussed, and an example of the object oriented design of the ontology will be given. In Section 7, some details of the implementation will be given. The evaluation and validation of RACAS will be discussed in Section 8, and, finally, in Section 9, some conclusions will be drawn and ideas for further work will be elaborated.

2 Approach

EUCLID RTP 6.1 started in 1993 with the objective to develop an innovative workstation for military C^3I. To meet this requirement, an agent-based approach for software engineering was selected. This selection was justified by the following foreseen benefits of agents:

- *A powerful metaphor for conceptualising complex systems*.
  The agent-based approach provides a powerful metaphor for software developers. It is quite natural to think of complex systems in terms of agents providing services.
- *An organisational approach to modelling organisational problems*.
  The ability to directly map real-world roles onto software components provides a powerful form of abstraction, especially in domains that are organisational in nature, such as the military domain.
- *Distribution of control*.
  Multi-agent architectures may provide distributed, heterogeneous computing environments in which multiple, concurrently-operating, intelligent agents exist. Agents may inter-operate in a seamless fashion, irrespective of where they exist within the environment.
- *The ability to support "plug-and-play"*.
  Due to a very loose coupling between agents, prototype applications may be developed quickly in order to experiment with ideas and algorithms, and later in the development cycle be replaced with minimal effect on the rest of the system.
- *Facilities for reuse*.
  The agents that reside in the agent community will typically be developed by many different people to address the needs of different applications or subsystems. If agents are well written and their capabilities made generic then they may be reused by many other applications.

At the start of EUCLID RTP 6.1, in 1993, the majority of agent applications were developed in an ad-hoc fashion because of a lack of a proven design methodology. Given the size of the consortium undertaking this project, a common approach was necessary in order to end up with applications capable of being integrated into a single workstation. The adopted approach to specification and design will be described in more detail in the following sections.

An additional problem was the lack of a multi-agent architecture capable of supporting real-time decision support applications. Already at the start of the project, the implementation of many different artificial intelligence techniques was foreseen. This implied that the resulting system would consist of a very heterogeneous collection of agents, which should nevertheless be capable of communicating with each other. Therefore, one of the first tasks of the consortium has been an ontological analysis of the C^3I domain resulting in the definition of concepts to be communicated between agents. The rather pragmatic approach to the construction of the ontology will be described in more detail in Section 6. A description of the multi-agent architecture that has been developed as a part of the EUCLID RTP 6.1 project can be found in [2].

3 Specification

The EUCLID RTP 6.1 approach to specifying functionality is called the Fun method [2]. The key concept in this approach is the Functional unit (Fun). A Fun is a specialised class of agent that is used to represent an organisation of agents. The main difference between a Fun and a (normal) agent is that a Fun has associated roles via which it can delegate tasks to its members. The roles of a Fun are played by agents which bring capabilities to the organisation in the form of tasks. Some of these agents may themselves be Funs representing subsidiary parts of the organisation.

In general, a Fun can do anything that a normal agent can do although typically the Fun will delegate most of its activities to its roles. In the minimalist case a Fun can be though of as an organisation's receptionist, passing on service requests to individual agents within the organisation. In a typical case a Fun can be thought of as a manager of an organisation, actively co-ordinating the activities of the agents playing roles within the organisation.

The specification of a Fun involves identifying the following concepts:

- *Objectives*, the desired goals towards which the Fun's tasks are directed;
- *Strategy*, for meeting its objectives;
- *Services*, provided to clients (end-users, or another Fun or agent); a service is the client's abstract view of the task(s) undertaken by the Fun;
- *Resources*, usually data stores which are owned by one agent (which may update it) but may be read by multiple other agents;
- *Roles*, that co-operate to provide the services.

In defining Funs an organisational analogy is recommended to ensure that the objectives, services, resources and roles are meaningful in the problem domain, and not just computing constructs. Criteria to break down functionality in Funs may be the following:

- *To use a Fun to model a human organisation in the real world.*

Real-world organisations have evolved throughout history to structures that have proven to be most efficient. Copying these structures in a system may aid to arrive at a better system architecture.

- *To provide an abstract architecture that is easy to understand.*
  An analogy with real-world organisations to analyse and describe system architectures will result in architectures, which are easier to understand, in particular for non-developers.
- *To enable 'plug-and-play' with agents.*
  Many different agents may implement the same role. Which agent actually takes on the role may be decided at run-time without affecting the rest of the system.
- *To facilitate reuse.*
  Common functionality may be organised in general Funs that may provide services to many other Funs. Examples of common functionality are message handling, file operations, and display.
- *To simplify distribution of functionality over a heterogeneous distributed computing environment.*

The Fun specification of RACAS has been made with emphasis on the latter three topics. The specification consists of one functional unit, RACAS itself, and one role within that functional unit, the role of an anytime, heuristic searcher. This specification allows a design in which the search algorithm will be in a separate agent. Such a design will create the possibility to make multiple implementations of a searcher agent to "plug-and-play" with these agents in order to compare search strategies. Furthermore, the search agent will not contain any code for user interaction, which will improve reusability in other applications. Finally, separating the searcher from the user interface will allow RACAS to run on a heterogeneous, distributed environment. The user interface could run on a lightweight computer, while the searcher could run on a powerful workstation.

4 System Design

The design approach adopted in the EUCLID RTP 6.1 project is agent-based at higher levels of abstraction, and object-oriented at lower levels. The main concept in system design is the concept of an agent. An application must be designed to consist of a number of autonomous, co-operative agents, which are together capable of solving the problem at hand. Agents perform tasks, and within these tasks, data is handled. For task and data design, an object oriented approach, Rumbaugh's well-known Object Modelling Technique OMT[11], has been used. Design patterns [4], such as the command pattern and the storable pattern, have been used where appropriate.
In adopting a multi-agent approach, an analogy with human organisations is made to represent the application at the high level. Uncertainty about how to partition the system at the high level may be resolved by appealing to the organisational analogy, or by appealing to software engineering rules-of-thumb, such as, in our case, separation of concerns.
In selecting agents, there are two goals. The first goal is to encapsulate a capability that can operate autonomously using a subset of relevant domain data and knowledge. The second goal is to ensure that this entity provides services that are relevant to the C^3I domain. If functionality is chunked into agents in a way that is inappropriate or too fine-grain, then the mapping onto the C^3I domain will fail.
Reuse has been the key design consideration during the design of the Resource Allocator for Close Air Support (RACAS). Experience has shown that, even when adopting an object oriented approach, reuse does not come naturally. A system has to be designed for reuse in order to be successfully reused. In the case of RACAS, design for reuse has been implemented by means of separation of concerns. The concerns identified are user interaction, domain knowledge, and search strategy. By dividing these concerns among three separate agents, these agents can be reused for roles in many other applications. For instance, a domain independent heuristic searcher will be capable of guiding search through the solution space of many problem domains. And an agent with knowledge of Close Air Support weapon systems and doctrine will be capable of serving applications other than resource allocation.
As a result, the design of RACAS (see Figure 1) identifies three co-operating agents:

- Agent Theresa (The resource allocator), which manages the application and the interaction with the user and with other applications;
- Agent Grap (General resource allocation planner), which is a domain-independent anytime heuristic searcher implementing a simulated annealing algorithm;
- Agent Deca (Domain expert on close air support and artillery fire support), which has knowledge of the military domain, and is capable of generating feasible solutions to the resource allocation problem.

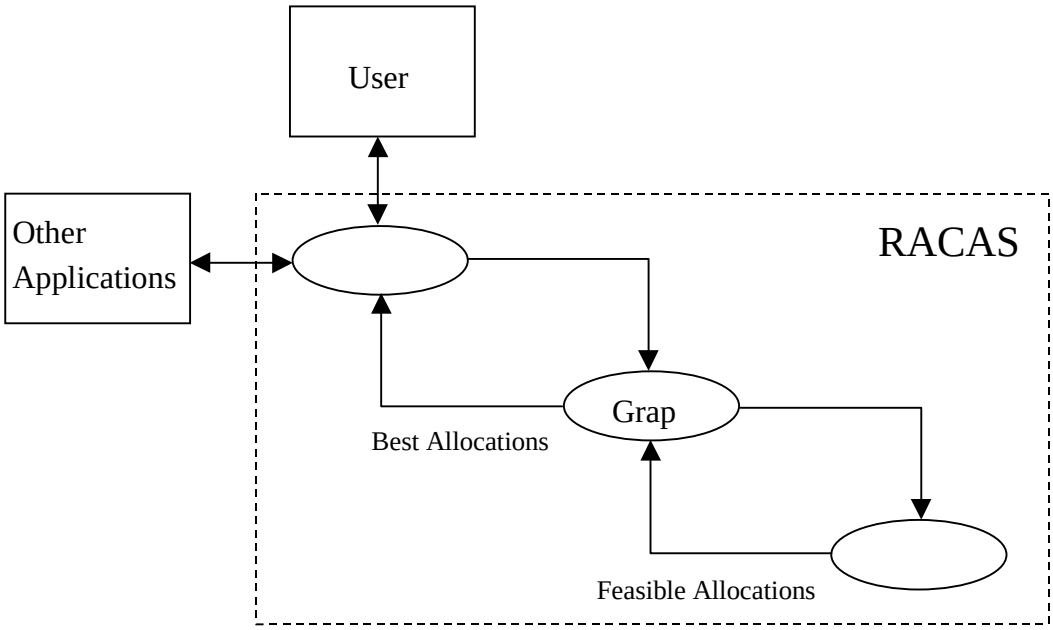

Figure 1 RACAS multi-agent system design

5 Agent Design

An agent design describes an agent by means of core activity, service activities, and internal activities to support the service activities. The core activity of an agent is the task that an agent executes indefinitely during its existence, for instance, sense its environment. Service activities are tasks performed by an agent to provide services to other agents. A service is defined by means of a number of entries by which the service may be called, and by means of a protocol, i.e. valid sequences of entry calls, to be used by client agents. Protocols have been specified by means of State Transition Diagrams. Internal activities are similar to service activities, but can only be called upon by the agent itself.
In the following subsections, the design of the three agents of RACAS will be described in more detail.

5.1 Agent Theresa

Agent Theresa is the agent playing the role of the Functional Unit (Fun). During its initialisation, agent Theresa must set up the required agent community by requesting from the architecture the population of its roles. The architecture contains a component, the agent manufacturer, which is responsible for finding agents that are capable of fulfilling the roles. In theory, any agent capable of fulfilling a role may be selected to fulfil that role. However, in the current implementation, only one agent exists for each role.
The core activity of agent Theresa consists of setting up the user interface and starting an internal activity to continuously monitor the environment for changes which affect the planning problem addressed by the RACAS application.

5.2 Agent Grap

Agent Grap takes on the role of the anytime, heuristic searcher as specified in Section 4. Therefore, agent Grap must have a service for providing best allocations using an anytime algorithm. This service is called Generate Resource Allocation Plans and has three entries:

- StartPlanning(aProblem): Initialises the domain expert agent Deca (see Section 5.3) with aProblem, and starts an internal activity which implements a simulated annealing algorithm;
- GetBestAllocationsSoFar(theNumberOfAllocationsRequested): Returns theNumberOfAllocationsRequested best allocations found so far by the simulated annealing algorithm;
- StopPlanning(): Stops the simulated annealing algorithm.

These entries must be called by a client agent in the order specified by the protocol in Figure 2. The internal activity Planning as shown in Figure 2, is the implementation of a simulated annealing algorithm. Simulated Annealing is based upon the physical process of annealing as applied to solids. Metropolis et al. [9] modelled this physical process with the introduction of an algorithm based on selective application of randomisation techniques (Monte Carlo techniques). Kirkpatrick et al. [7] and Cerny [1] first applied this algorithm to combinatorial optimisation problems.

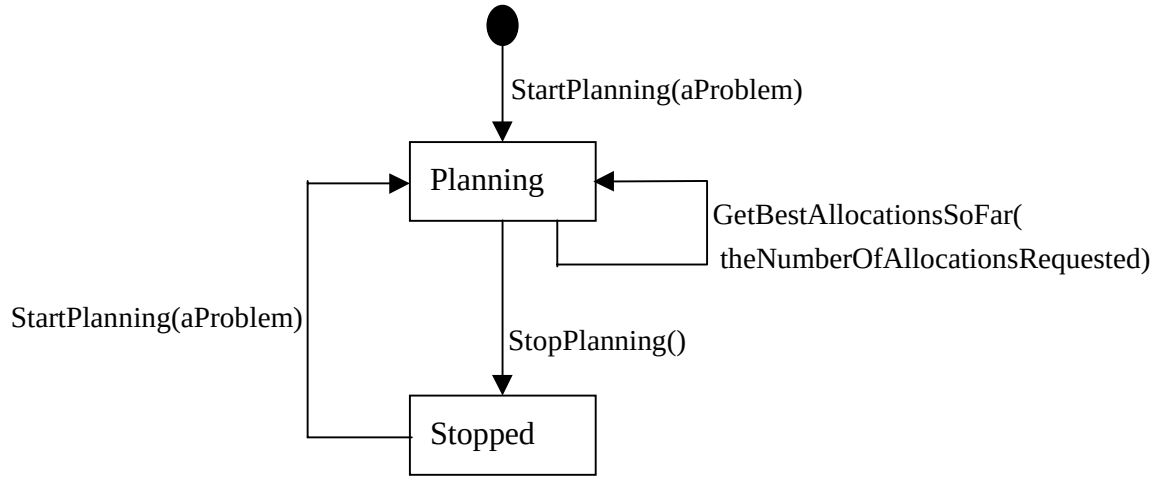

Figure 2 Protocol for service Generate Resource Allocation Plans

The Simulated Annealing seeks near-optimal solutions to a problem at a reasonable computational cost, with no guarantee of optimality, but with a "best-so-far" solution available at any time. The Simulated Annealing technique has proven to provide robust and flexible algorithms which can be applied, with only very minor changes, to a wide spectrum of problems [e.g. 12]. These algorithms require only a method for determining the quality of a solution and a method to move between solutions, both of which are relatively simple tasks for the majority of problem types. For the problem described in this paper, these methods are implemented by Agent Deca (see Section 5.3). Simulated Annealing control parameters have been determined empirically. No effort has been allocated to formally prove correctness and convergence of the algorithm. For randomly generated problems, the implemented algorithm converges to a good quality solution in finite time.

The implemented algorithm is as follows:

1. Initialise control parameters.
2. Ask Agent Deca for an arbitrary solution (including a quality attribute) and store this solution as the best solution so far, and as the current solution to continue search from.
3. Ask Agent Deca for a neighbour solution of the current solution.
   3a. If the neighbour has a better quality, then neighbour becomes current solution to search from, and if neighbour is better than best solution so far, store neighbour as best solution.
   3b. If quality of neighbour is equal, or worse, then:
      3b1. Determine probability that neighbour will nevertheless be better solution to continue search from. This probability is determined by a number of algorithm control parameters and declines every iteration of the algorithm. The tendency to continue searching from a worse solution therefore decreases as the search progresses (the possibility to continue search from a worse solution has been introduced to avoid local optima).
      3b2. If the probability is high enough, replace current solution by neighbour, else increase the number of failed iterations.
4. Repeat Step 3 until the number of failed iterations exceeds the maximum number of failed iterations allowed.

5.3 Agent Deca

In order to apply the simulated annealing algorithm of agent Grap to the resource allocation problem described in Section 3, Agent Deca has to implement a service, called Provide Resource Allocation Domain Knowledge, which has the following six entries:

- InitialiseProblem(aProblem): Initialises the service with the Close Air Support and Artillery assets available for allocation, and the targets to which these must be allocated;
- RandomSolution(): Implements the initial solution function as described above;
- RandomNeighbour(aSolution): Implements the neighbour function as described above;
- SolutionSpaceSize(): Returns an estimate for the size of the solution space;
- MinimumScore(): Returns an estimate for the quality of the worst possible solution;
- MaximumScore(): Returns an estimate for the quality of the best possible solution.

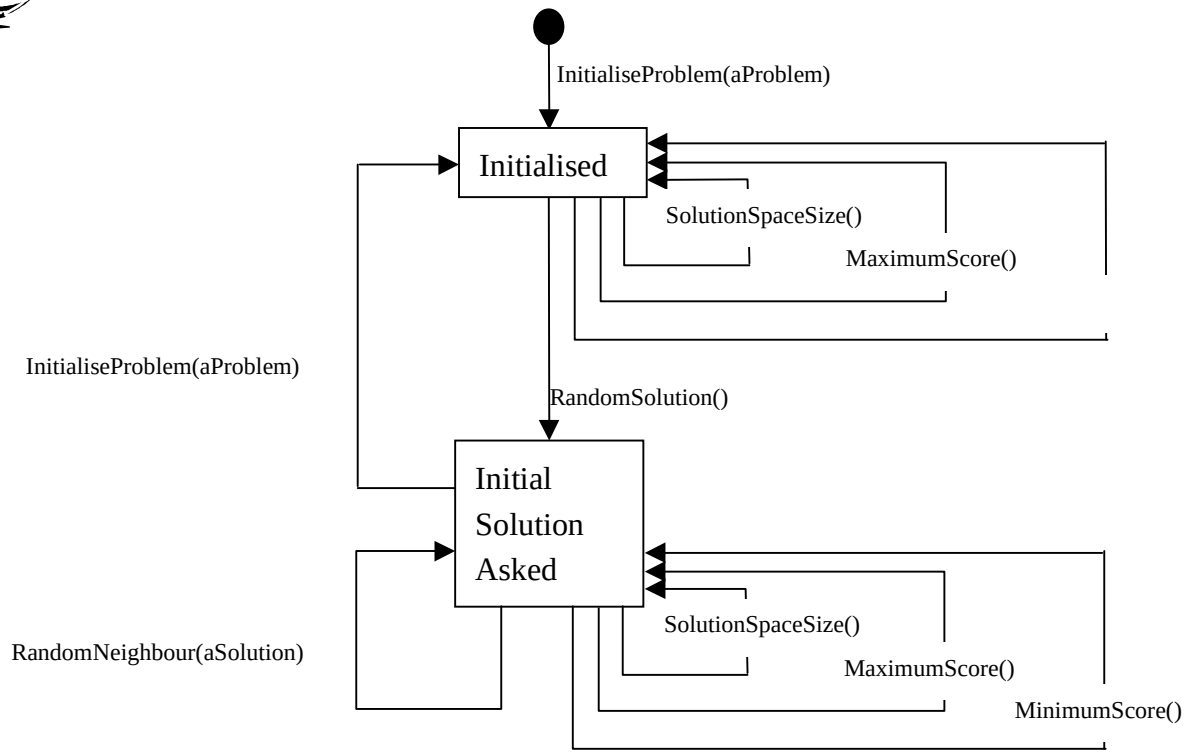

Figure 3 Protocol for service Provide Resource Allocation Domain Knowledge

These entries must be called by a client agent in the order specified by the protocol in Figure 3.
The functions required by the above entries are functions to generate an initial solution, to generate a neighbour solution, and to calculate the quality of a solution. These functions have been implemented in the following way.
To find an initial solution, for each asset, a target is selected by traversing a randomly shuffled list of targets, until a target is found that does not violate any domain constraints. In the current implementation, these constraints are:

- The target must be in range of the asset;
- The target must not yet be allocated to the required extent;
- The asset must have weapons capable of damaging the target.

If such a target cannot be found, the asset is not allocated to any target (which is considered a valid option).
A neighbour solution is generated from a given solution by randomly selecting an asset and allocating that asset to a different target satisfying the domain constraints.
The quality of a solution is determined by the expected damage to the targets. For each target, the expected damage is calculated based on the types and amount of weapons used against the target, and the effectiveness value of the weapon types against the target type. These effectiveness values are parts of the domain knowledge. The expected damage is multiplied by the target priority to give a target score. The quality of a solution is the sum of all target scores.

6 Ontology

An essential element of a multi-agent system is the language (ontology) that the agents use to communicate. Since the agents represent autonomous capabilities in the C^3I domain, their language should consist of accepted terminology in the military C^3I domain. At the higher level this corresponds to a defined use of English. At the lower levels this corresponds to a shared interpretation of objects and services.
The definition of ontology adopted within the EUCLID RTP 6.1 project is similar to the definition of Gruber [6]. In the project standards, ontology has been defined as follows:
Ontology defines the basic terms and relations comprising the vocabulary of a topic area, as well as the rules for combining terms and relations to define extensions to the vocabulary with the aim of establishing the ground rules for modelling in a domain.
Within EUCLID RTP 6.1, the approach towards constructing the ontology has been rather pragmatic. Only a partial ontology for the military C^3I domain, called the Grace Common Model, which describes the general, high-level concepts of the C^3I domain, has been constructed. RACAS, as each application in EUCLID RTP 6.1, has developed its own task-specific extension to this common model.

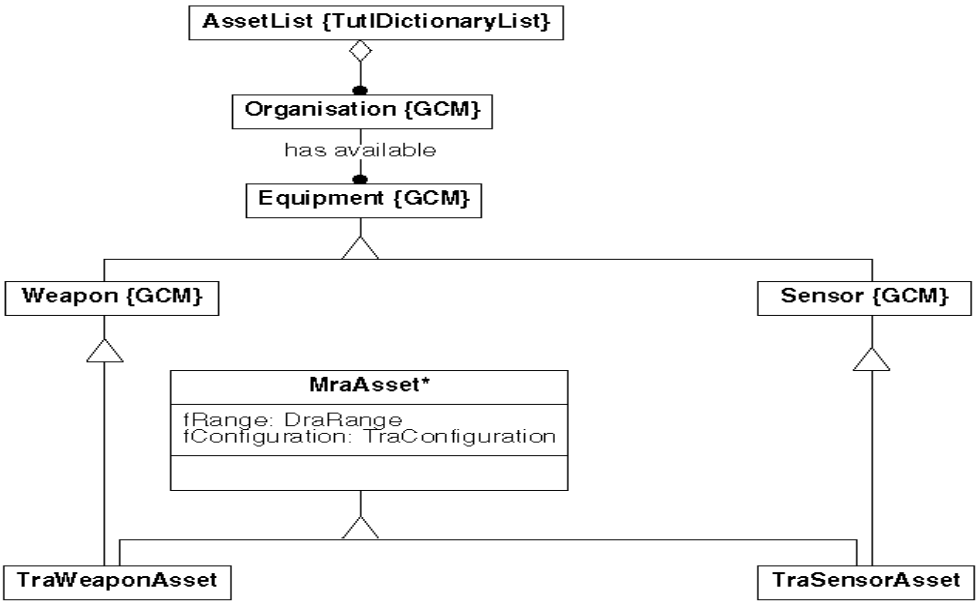

Figure 4 Example of an ontology object model

A possible procedure for extracting ontology [3] begins by analysing a paper knowledge base that describes the task domain in English. In the context of EUCLID RTP 6.1, the ATCCIS (Allied Tactical Command and Control Information System) model for generic command and control, and NATO Allied Tactical Procedures have served as a basis for the ontology. In addition to paper knowledge bases, project internal as well as external domain experts have provided valuable inputs to the ontology.
During the design of the Resource Allocator for Close Air Support, the ontology has been translated into an object-oriented design. An example object model is shown in Figure 4. In this figure, the concept of asset is defined. An asset may be either a weapon or a sensor, which have been defined in the common part of the ontology, the Grace Common Model (GCM). Therefore, two additional concepts, weapon asset and sensor asset, have been defined. These concepts inherit the characteristics of both the GCM concepts and the asset concept (range and configuration). The prefixes Mra- and Tra- are specific to RACAS concepts and have been introduced to avoid name clashes with other applications, which may also define a concept such as asset, but with a different meaning.

7 Implementation

RACAS has been implemented on top of the programme's multi-agent architecture Cable[2]. This architecture has provided the agent developers with an Agent Definition Language, which contains keywords to define an agent's memory (i.e. objects available to all tasks and services of the agent), tasks, and services. A pre-processor translates these definitions to C++ classes, which hide multiple thread management and communication through Orbix, a commercial implementation of the Common Object Request Broker Architecture CORBA[10]. The agent services and tasks, as well as the ontology objects have been implemented in C++ directly. The User Interface of the application has been developed with Ilog Views, a commercial tool for designing user interfaces, and implemented in C++ as an applet, similar to the concept of applets in Java.

8 Evaluation and Validation

For quantitative assessment of RACAS, 24 test runs have been made on a Sun Ultra2 Creator workstation. Each test run was at least one-and-a-half hours long, with three hours being the longest run. The problems submitted to RACAS have been extracted from the project's scenario. Therefore, the maximum number of targets during a test run has been 20.
The size of the problems is in the order of $(T+1)^A$, in which T is the number of targets and A is the number of assets. Exhaustive search for the optimal solution to the submitted problems is impractical. As a result, an absolute measurement of the quality of the solutions found by RACAS cannot be given. Therefore, during evaluation, data showing how fast the simulated annealing algorithm of RACAS converges have been gathered. During each run, the time of the first solution, quality of the first solution, quality after one minute of search, and after five and ten minutes of search, have been measured. The qualities have afterwards been measured against the best solution found after one- and-a-half hours of search. Please note that this solution is not necessarily the best solution possible. The results are in Table 1.

Table 1: Evaluation results for RACAS

Assets	Targets	Time 1st (seconds)	Quality 1st (%)	Quality 1 min (%)	Quality 5 min (%)	Quality 10 min (%)
12	10	0.65	48	88	100	100
12	20	0.63	63	97	98	100
48	10	0.70	36	79	91	91
48	20	0.55	19	75	88	96
84	10	0.75	35	71	87	89
84	20	0.90	13	56	87	96

The results in Table 1 show that, even for large problems, the simulated annealing algorithm converges quickly. Within ten minutes, the quality is already over 90% of the quality after one-and-a-half hours.
RACAS has been demonstrated to military experts at several occasions. The overall assessment of the tool is very positive. Because of the automation of lower level tasks of the allocation process, less time is needed to find feasible allocations. As a result, more time is left to select an appropriate allocation to the situation at hand and work out the details.
An operational validation of RACAS has not been performed yet. EUCLID RTP 6.1 is an "open" project and therefore does not use classified military information. As a result, the domain models have been simplified to simulate operational application, and need adaptation for operational use. The feasibility of the approach, however, has been demonstrated.

9 Conclusions and Further Work

In this paper, the development of an agent-based, time-critical decision support tool in the field of Command, Control, Communications, and Intelligence, has been described. This tool, called the Resource Allocator for Close Air Support (RACAS), applies a simulated annealing algorithm to the problem of allocating tactical air resources to close air support missions. The tool has been generalised to also allocate tactical air resources to battlefield air interdiction missions and to allocate artillery assets to a number of prioritised targets.
For specification, the Fun method has been used. This method has proven to be elegant and useful on the higher levels of abstraction. However, since the lowest level of detail in the Fun specification is the task, some essential analysis information is missing. Therefore, the transition from analysis to design is difficult, especially when documentation is the only source of information to the design team.
For the design of RACAS, techniques from agent-based computing, knowledge engineering (ontology), object-oriented design, and design patterns have been combined. Reuse has been the key design consideration from the start. This design decision has been implemented by applying a separation-of-concerns guideline. The result is a design with three agents: one for communication with the user and application external agents, one for executing the simulated annealing algorithm, and one for providing the necessary domain knowledge. An early benefit of this approach is the possibility to run the interface agent on a light-weight PC, while the computational agents run on a Sun Ultra 2 Workstation.
The ontology has been designed using Rumbaugh's Object Modeling Technique, and implemented in C++. The ontology work has contributed to an important extent to the design for reuse of RACAS. The ontology describes the problem-specific knowledge, independently from the proposed solution (anytime simulated annealing algorithm). By implementing the ontology in one agent, and the search algorithm in another, an explicit division has been made between problem and solution. This explicit division should make reuse of both agents in other applications much easier.
RACAS has been implemented in C++ on a Sun UltraSparc with Solaris 2.5, and successfully demonstrated on several occasions. The application consists of three co-operating agents running on a project-specific, multi-agent architecture, called Cable, which uses a commercial-of-the-shelf CORBA implementation (Orbix) as the underlying middle-ware. The user interface of the application has been developed with Ilog Views, and implemented in C++ as an applet, similar to the concept of applets in Java.
RACAS will be integrated in the NLR Command and Control Facility, which is currently under development. This facility will bring together the products of various projects. These products include advanced situational awareness, multi-sensor data fusion, adaptive planning, mission generation, and mission monitoring tools.
Research into real-time planning and re-planning will continue both in the military and in the civil domain. The complete demonstrator of EUCLID

RTP 6.1, including the multi-agent architecture Cable, will be installed at NLR to serve as an experimentation facility for agent applications. The NLR is also looking for opportunities to combine real-time planning and intelligent agents in the European Community's Fifth Framework Programme.

10 References

[1] Cerny, V., Thermodynamical Approach to the Travelling Salesman Problem: An Efficient Simulation Algorithm, in: Journal of Optimization Theory and Applications, 45(1), pp. 41-51, 1985.

[2] Dee, C., Millington, P., Walls, B., Ward, T., Cable: A Multi-Agent Architecture to Support Military Command and Control, submitted to the 1998 Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM).

[3] Freiling, M.J., Alexander, J.H., Messick, S.L., Rehfuss, S., Shulman, S., Starting a Knowledge Engineering Project – A Step-by-Step Approach, in: AI Magazine, 6(3), Fall 1985.

[4] Gamma, E., Helm, R., Johnson, R., Vlissides, J., Design Patterns – Elements of Reusable Object Oriented Software, Addison-Wesley, 1995.

[5] GRACE home page, http://public.logica.com/~grace/

[6] Gruber, T.R., A Translation Approach to Portable Ontologies, in: Knowledge Acquisition, 5(2), pp. 199-220, 1993.

[7] Kirkpatrick, S. Jr, Gelatt, C.D., Vecchi, M.P., Optimization by Simulated Annealing, in: Science, 220(4598), pp. 671-679, 1983.

[8] Martin, P., EUCLID RTP 6.1 – The Application of Artificial Intelligence Techniques to C^3I Workstations – Introduction, Initial Results and Lessons Learned, EUCLID Symposium, Paris, France, October 1996.

[9] Metropolis, N., Rosenbluth, A.W., Rosenbluth, M.N., Teller, A.H., Teller, E., Equation of State Calculations by Fast Computing Machines, in: The Journal of Chemical Physics, 21(6), pp. 1087-1092, 1955.

[10] Object Management Group, The Common Object Request Broker: Architecture and Specification, Revision.2.0, July 1995.

[11] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W., Object Oriented Modeling and Design, Prentice Hall, 1991.

[12] Telfar, G., Generally Applicable Heuristics for Global Optimisation: An Investigation of Algorithm Performance for the Euclidean Traveling Salesman Problem, Master's Thesis, Victoria University of Wellington, 1994.