Sensor scheduling using Intelligent Agents

René Thaens
Delft University of Technology
Mekelweg 4
P.O. Box 5031 / 2600 GA
Delft, The Netherlands
rene.thaens@wxs.nl

Abstract – *As a result of the versatility of modern phased array radar systems and the complexity of the modern battlefield, it is necessary to design planning algorithms that allow the full exploitation of the sensor's capabilities. Optimising the radar's performance is shown to belong to the class of NP-complete problems, thereby excluding the posibility to find an algorithm capable of providing solutions in polynomial time or shorter. Based on this hypothesis, alternative approaches for designing the scheduler need to be explored. This paper presents an innovative radar scheduler based on an object-oriented method of generating tasks, which are prioritised by a Multi-Agent system. The resulting planning provides a near-optimum allocation of sensor time and energy to all maintenance tasks required for the Air Picture. The paper concludes with the presentation of a basic IA task scheduler and the key performance issues that will be addressed in the near future.*

Keywords: Radar, scheduling, multi-agent system, object oriented task generation.

1 Introduction

Since 2002, Delft University of Technology (DUT), Thales Naval Nederland and the Royal Netherlands Naval Academy (RNLNC - KIM) have conducted a joint research project called STATOR (Sensor Tracking and Tuning on Object Request) which is aimed at managing a suite of sensors based on operational and technical performance parameters. One of the research topics within the STATOR project is the design and implementation of a sensor task scheduler capable of allocating sensor time and energy to the tasks required for maintenance of an air picture. The resulting schedule is aimed to be optimal with regard to the quality of the air picture as derived from the prevailing operational needs. The overall goal of STATOR is to find appropriate management approaches oriented towards operational measures of performance.

Although the STATOR project will eventually focus on a suite of sensors, the initial research is limited to a single multi-function phased array radar (MFR). A radar of this type enables instantaneous (re-)direction of the radar beam and considerable flexibility in the choice of beam parameters. These capabilities and the number of modes in which the radar can be operated are essentially too complex to be managed manually. Many previous attempts to design a control loop for such a complex process are based on technical measures of performance instead of operational parameters. In the case of STATOR, the operational relevance of the sensor data is the parameter to be optimized.

The purpose of this paper is to present a new architecture for generating and scheduling radar tasks for a multi function radar. In contrast with currently known radar scheduling algorithms, our architecture allows for the inclusion of both operational and technical inputs in order to optimize the radar performance.

This paper starts with the introduction of the generic radar scheduling problem, belonging to the class of NP-complete problems. As a next step, an innovative architecture is presented, based on intelligent agent technology. Supporting this architecture, a simplified Intelligent Agent scheduler is described, followed by some closing remarks on future extensions of our model.

2 The Radar Scheduling problem

2.1 Introduction : typical MFR tasks

The primary goal of the MFR onboard a naval vessel is to provide a complete, accurate and reliable air picture to the operator. In addition, the MFR can assist in cueing onboard sensors and weapon systems when engaging targets. In order to do so, the MFR needs to revisit objects already identified and in track, and also has to search for new objects. Due to the characteristics of phased array radar, such as electronic beam steering and forming, the radar beam can switch almost instantaneously between two directions in space. The

'track' and 'search' tasks are therefore virtually decoupled and can be scheduled consecutively to, but independent from, other tasks. This is one of the main advantages of MFR over classic radars with revolving antennas, which generally provide fixed update rates for revisiting objects in the coverage area [1]. This section presents a generic radar scheduling problem (RSP) based on a series of tasks that need to be planned by the radar scheduler.

Radar tasks like track updates and search requests are generated via an object-oriented analysis of the most recent air picture. Detected targets are instantiated via a generic class of targets described by a variety of attributes like state vector, recent track history, identity, classification etc. Attached to these attributes are measures of quality indicating the reliability and accuracy of the specific attribute assignments. After track initialization, most of these attributes will be undefined but they must be addressed by the MFR as time goes by. Furthermore, some of these attributes will change over time, for example state vector attributes etc. In an object-oriented approach, every instantiation continuously evaluates its own attributes to determine when updates are required. The need for updates on attributes is translated into update requests to the radar system. It is possible that a single object generates a variety of requests in parallel, e.g. an update of state vector attributes together with a request for a high range resolution profile measurement as input to a non-cooperative target recognition system (NCTR) to determine the type of the target.

Search tasks can be treated in a similar manner. The total coverage area is subdivided in smaller areas (search cells) that have to be searched for new targets. As in the case of track updates, every search cell is capable of generating search tasks. As long as no object is detected, new search tasks for the cell are created. Once an object is detected in the search cell, a track is initiated resulting in the generation of both track update tasks for the object and search tasks for the search cell. Fig. 1 provides a schematic overview of this object oriented approach to search and track task generation.

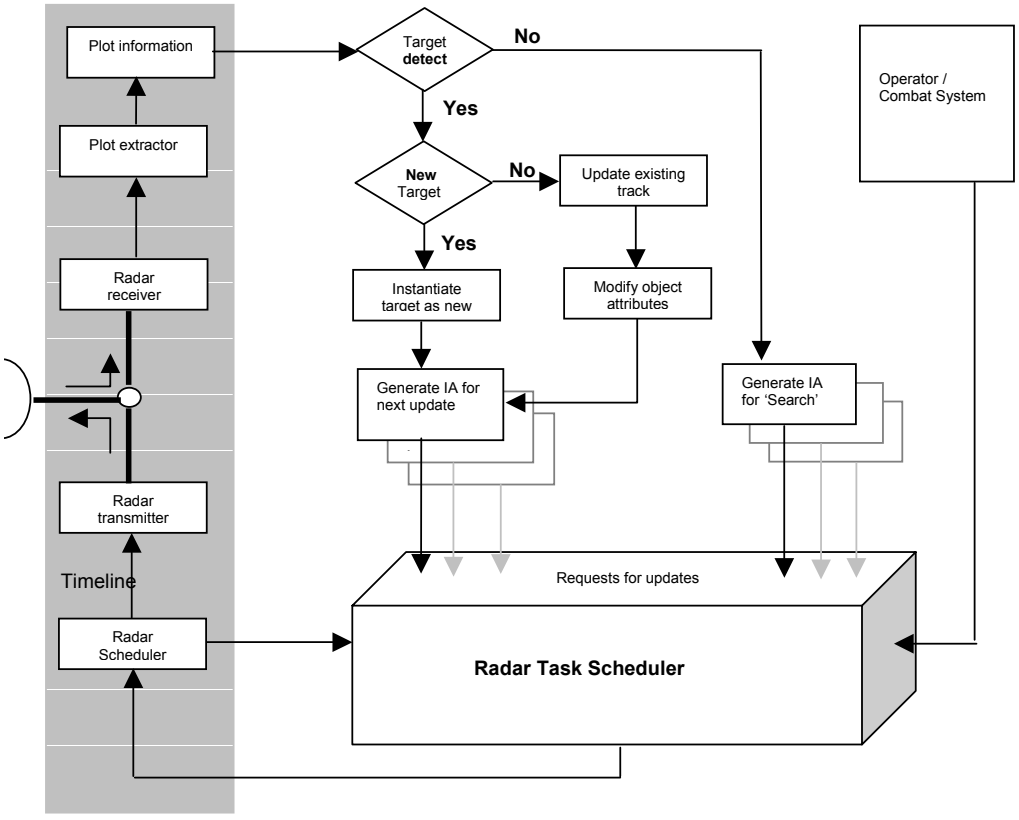

Fig. 1 Object Oriented search and track generation

2.2 Description of the generic scheduling problem

The RSP concerns the planning of N tasks for future execution by a sensor. A task T_i, $i \in \{1, 2, \ldots, N\}$ is defined as:

T_i = { D = duration
t_s = earliest start time
t_f = latest completion time
P = priority of this task, $0 \leq P \leq 1$
}

with $D < t_f - t_s$, according to fig. 2.

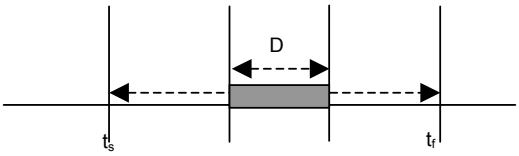

Fig. 2 Time variance of a single task

The measure P is an indication of the absolute importance of this particular task. In case of radar scheduling, P can be derived from an operational risk analysis based on parameters like the identity of the object, flight profile, Rules of Engagement (RoE) in force, etc. For example, a high-flying airliner flying away from the sensor platform will most likely have a lower priority than an inbound low-flying unidentified fighter, based on the potential risk both targets may cause in the operational scenario.

STATOR deals with the definition of a feedback loop from operational scenario to sensor management module in which Operational Measures of Performance (OMOPs) are used to prioritize radar tasks. OMOPs are formed by the set of quantifiable measures by which the mission success of the ship is expressed. Similarly, Technical Measures of Performance (TMOPs) are used to express the performance of the suite of sensors. The suite of sensors is managed with respect to maximizing the OMOPs but within the constraints set by the sensors' TMOPs. The actual design of such a control loop is also a task within the STATOR project.

Once the scheduler has placed a set of tasks on a timeline, the quality of this placement needs to be determined. As the measure of quality for a certain placement, a Utility function U is defined:

$$U = \frac{\sum_{i=1}^{N} \delta_i \cdot P_i}{\sum_{i=1}^{N} P_i} = \frac{\sum_{i=1}^{N} \delta_i \cdot P_i}{U_{\max}} \qquad (1)$$

with $\delta_i = \begin{cases} 1 & \text{if task in planning,} \\ 0 & \text{else.} \end{cases}$

Assuming a single sensor configuration, the radar system is capable of executing a single task at a time. Tasks are *non-preemptive*, meaning that they have to be completed once started [2]. Due to the fact that the task priority is derived from the operational measures of performance, the aim of radar task scheduling is to find planning solutions for which the summed priority is maximized.

2.3 Complexity of RSP

Finding the optimal planning for a given problem requires an amount of processing time that is related to the number of processor steps required. In order to classify this amount of time, the time complexity function is used [2]. The time complexity function provides an asymptotic upper bound for the calculation complexity of a function f with domain n = {0, 1,…N} via:

$$f(n) \in O(g(n)) \text{ iff } \exists_{c>0} \forall_{n \geq 0} \quad |f(n)| \leq c|g(n)| \qquad (2)$$

The order of magnitude of the time complexity function determines the class of the problem complexity. A polynomial time algorithm has a time complexity function *O(p(n))* for some polynomial function p with n samples. Exponential time algorithms cannot be bound by a certain polynomial function and will have a tendency to explosive growth for larger problem instances. The complexity class to which a problem belongs, provides an indication of the difficulty of finding potential solutions.

To find the complexity class to which the RS problem belongs, we will use the similarity of our problem to the so-called 'Knapsack Problem' (KSP). KSP is an already proven NP-complete problem as described in [2]. One method to prove NP-completeness of a problem Π is to find a (polynomial) transformation that maps a known NP-complete problem Π' to Π. KSP is defined as:

Instance : A finite set U, a 'size' $s(u) \in Z^+$ and a 'value' $v(u) \in Z^+$ for each $u \in U$, a size constraint $B \in Z^+$ and a value goal $K \in Z^+$.

Question : Is there a subset $U' \subseteq U$ such that

$$\sum_{u \in U'} s(u) \leq B \quad \text{and} \quad \sum_{u \in U'} v(u) \geq K \qquad (3)$$

The name 'knapsack' refers to the more popular definition of this problem, in which the knapsack is a bag of given volume. This bag must be filled with items of certain sizes and weights, such that the total knapsack weight is maximized.

The proof for NP-completeness of KSP is based on 'proof by restriction', which requires showing that Π contains a known NP-complete problem Π' as special case. For further details on the proof of KSP ∈ NP-complete, the reader is referred to [2].

As in the case of KSP, RSP deals with selecting a subset of predefined units called 'tasks' under a constraint (available time in RSP, the size B of the Knapsack) while satisfying an optimization function. The major difference between RSP and KSP is the time dependency of RSP tasks. This time dependency originates from the planning freedom given by the duration $D \leq (t_s - t_f)$, meaning the task can be scheduled anywhere on the interval t_s to t_f. The time dependency of RSP can be eliminated by replacing the original time flexible task by a set of sub-tasks representing all planning options per task. In such a way, the original time flexible task set is expanded to

$$M = \prod_{i=1}^{N} \left((t_{f_i} - t_{s_i}) + 1 - D_i \right) \quad (4)$$

new tasks, as demonstrated in fig. 3.

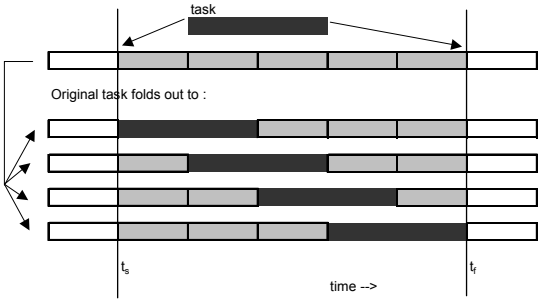

Fig. 3 Single task expansion to eliminate time dependency

These new tasks are fixed in time and do not have any planning freedom left, meaning that the original RSP is transformed into M independent KSP's. Solving the RSP is therefore equivalent to solving the stack of M KSP's followed by a MAX operation on the M results in order to find the optimum planning. Based on the proven NP-completeness of KSP, it is concluded that the RSP is NP-complete.

2.4 Consequences of NP-completeness

The object-oriented approach results in a steady flow of requests for time and energy to be provided by a single sensor or even a suite of sensors. In complex scenarios the number of requests can increase dramatically, to such levels that the available sensor time is not sufficient to service the entire set of requests. The radar task scheduler needs to optimize the set of planned tasks on the timeline with respect to the resulting quality of the air picture. We have shown the RSP to belong to the category of NP-complete problems, thereby preventing the construction of an algorithm that provides the optimum schedule under all problem instances in finite time. Due to this consequence of NP-completeness, the algorithm for a radar scheduler needs to provide a solution for the planning problem that is 'as good as possible'.

The prime consequence of classifying RSP as NP-complete is that no polynomial time algorithm is available for solving the problem. Workable solutions are only possible based on approximations of the RSP. In the course of history, a wide variety of approximations has been proposed for NP-complete problems. These approximations range from brute force computations bounded by available processor time to more intelligent approaches based on artificial intelligent applications. Given the fact that the RSP is part of the overall STATOR research, the following requirements are used to evaluate candidate approximations:

- Accuracy: approximation needs to provide accurate solutions to the NP-complete RSP
- Timeliness: solutions need to be available within 'reasonable' time, where 'reasonable' is determined by the actual available time for the radar scheduler
- Convergence: the longer the system calculates on the approximation, the better the solution needs to be
- Flexibility: the radar scheduler needs to be able to deal with uncertain and incomplete information (meta knowledge)
- Learning: the approximation should be able to improve its performance with increasing time of operation
- Conformity: the implementation of the scheduler should be compliant with the object oriented methods of search and track task generation
- Transparency: after reaching a solution, the system must be capable of explaining the way this solution is achieved to the human user

Although not fully quantifiable, these criteria define a baseline of preferred properties for the identification of an approximation method from the family of intelligent systems like expert systems (ES), neural networks (NN), genetic algorithms (GA) and multi-agent systems (MAS). Due to their lack of transparency, NN's are excluded from further research. In addition, ES and other rule-based approaches do require an extensive knowledge of the problem domain, inhibiting the flexibility to deal with uncertain and incomplete information. Literature has described various applications of MAS for planning purposes under complex circumstances [3, 4]. As MAS seem to be able to fulfill the previously listed requirements for solving RSP, we have selected the intelligent agent approach for designing a radar scheduler. Only the matter of timeliness of applying MAS to RSP is yet an unsolved issue. This will be addressed briefly in the final section and chapter 3 of this paper.

2.5 Motivation for intelligent search

The basis for the STATOR research project is the desire to design a sensor tuning loop via which information from the sensors is used to optimize sensor performance with regard to the sensor output. This loop is based on an object oriented task generation process, which produces a wide variety of tasks needed to maintain the quality of the air picture. The number of these tasks is likely to extend above the saturation level of the radar, meaning that the total time and timeliness for all tasks exceeds the available radar resources. This implies that the radar scheduler has to focus only on the most important tasks that can be accomplished in the available time. Determination of which subset of tasks can be identified as 'most important' requires knowledge on a level higher than task level. Actual and expected scenario developments like operational procedures and the technical state of the radar need to be included in the decision process.

All information above task level is considered to be meta-knowledge for the radar task scheduler. Current radar systems employed in both civil and military applications do not take this meta-knowledge into account. Scheduling tasks based on the underlying NP-completeness of the problem and the need to include potentially incomplete or even contradictory meta-knowledge requires some sort of intelligence to provide workable solutions. In addition, the radar scheduler must be capable of optimizing the timeline for a dynamic network of sensors in which the number and quality of sensors varies over time. Both the need for inclusion of meta-knowledge and the requirement for dynamic network configuration call for the application of intelligent approaches to solving RSP.

3 IA Scheduler architecture

3.1 Intelligent Agents (IA)

Although the current literature still lacks an all-encompassing definition of an 'intelligent agent', some agreement has been reached on the following properties of these entities:

- An 'agent' resides in its environment from which it receives inputs via a sensory device and acts so as to affect that environment in some way through effectors [5].

- Agents are autonomous goal-driven problem solvers able to communicate with other agents in their environment and to modify their behavior as a function of a changing environment.

Both descriptions of intelligent agents center on the theme of being able to communicate with other agents in order to solve a problem cooperatively. Agents can behave reactively or pro-actively and can be self-interested or cooperative, depending on how the problem is modeled by the system designer. Furthermore, agents can be programmed in such a way that their behavior can be modified according to previous success or failure in reaching a specific goal. Thus, a certain level of learning capability is possible, thereby exhibiting the intelligent behavior required to deal with the particularities of RSP.

3.2 Multi-Agent Systems

Multi-agent systems are as loosely defined as the 'intelligent agent' itself. In general, MAS are typically distributed systems in which several distinct components, each of which is an independent problem-solving agent, come together to form a coherent whole [5]. The MAS provides agent life cycle management and describes the communication, coordination and cooperation protocols to which the agents have to adhere. In doing so, a MAS does not necessarily provide opportunities that could not be reached before, but does allow for the inclusion of various sources of information in order to derive a solution.

3.3 Applying a MAS to the RSP

Previous applications of MAS in production environments have shown the capability of these systems to comply with the general requirements as laid down in section 2.3 [5]. A MAS is a logical extension of the object oriented approach for generating search and track tasks for the radar. Every generated task can be instantiated by an agent with the single goal of claiming time from the available radar system or systems. The radar in its turn submits agents representing available time to a general market place. On this market both types of agents meet and are able to communicate. The result of this communication is the allocation of available radar time to a prioritized set of search and track tasks. Fig. 4 depicts this market situation with both the supply and demand agents.

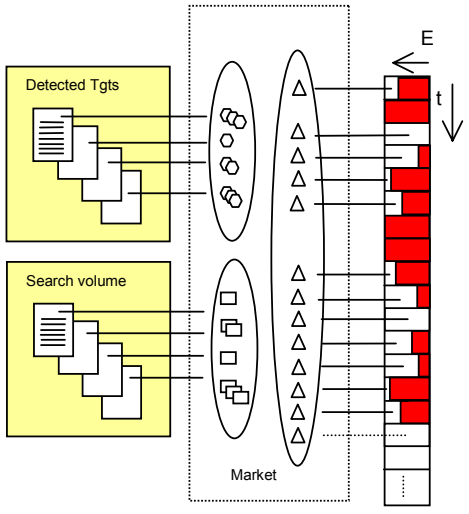

Fig. 4 Supply and demand agents on marketplace

According to the definition of IA's, these agents can be relatively simple and do not necessarily have to know

about the existence of the entire set of other agents. The success of such a scheduler depends heavily on the coordination and cooperation mechanisms via which the agents communicate. The negotiation protocols can be based on either heuristic theory of bargaining or the formal game theoretic approach [6]. Both approaches do have iterative processes in common via which a system of offers and counteroffers should lead to an acceptable overall result for all agents involved.

This basic architecture only caters to a mapping of tasks on available timeslots, but does not include any feedback on developments in the operational scenario or on constraints laid down by the technical performance of the radar. These feedback loops based on OMOPs and TMOPs are included in the MAS architecture by introduction of two separate sets of agents. Currently for every target-representing agent the initial priority is given at the moment of creation. This is an artificial number, which is derived from the class to which the agents assumingly belongs. For example, fighters will have higher initial priorities than slow moving air transport aircraft due to their inherent mobility and potential threat to own forces in the scenario.

Thus the initial priority assignment per task is not depending on detailed operational inputs. In reality however, the priority of a specific task is scenario dependent, which requires the agent to be cognizant of its environment. In our model, this dependency is incorporated, not by extending the agent model, but by the use of dedicated agents. Modifications in the scenario like changing 'rules of engagement' etc can be reflected by introducing a 'scenario agent', which engages in modifying the existing priorities of agents on the marketplace. In a similar way, dedicated agents can represent limitations imposed by the technical constraints of the radar system, such as required duty cycles etc.

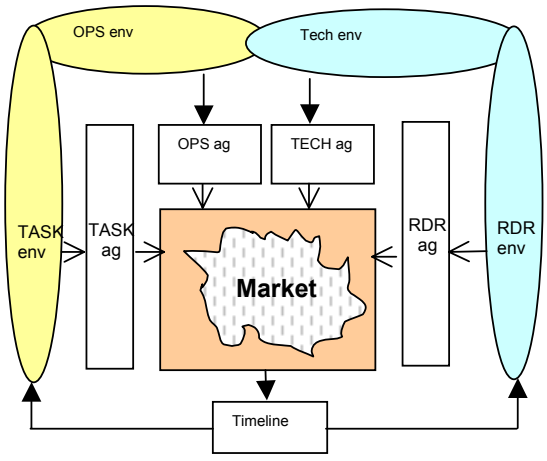

Fig.5 Sources of agents

Fig. 5 shows how operational and technical feedback is incorporated into the planning process at agent level. This figure shows the central marketplace surrounded by four agent spawning processes: TASK agents from the operational environment, RDR agents from the available timeline, OPS agents representing the operational innovations and finally TECH agents derived from technical constraints. As mentioned earlier, every agent spawning process is based on observations of a certain environment and a set of rules governing the instantiation of agents. Within STATOR the focus is currently on the generation of OPS agents rather than TECH agents.

A short example illustrates the role of the OPS and TECH agents on the market place: consider an operational scenario with a low threat level and two objects within the coverage area of the radar. One of these objects is approaching the radar at fairly high speed and low altitude. The second object is at cruising level steering away from the radar. The identity of the objects is not yet known, although it has been determined previously that object 1 is a high maneuverable target. In a low threat scenario, the radar needs to allocate time to both targets because the objective is to provide an accurate and reliable air picture, which includes both targets. Due to the low threat level, the scheduler will consider both targets to be equally important. Once the scenario changes from a low to a high threat level, the radar scheduler needs to recognize the fact that object 1 must have priority over object 2 because most likely the actual threat of object 1 is higher. An OPS agent, aware of the new operational environment, communicates with both TASK agents in order to bias their already preset but outdated priorities. Biasing the priority means raising the priority of the object 1 TASK agent with respect to the object 2 TASK. Thus, the radar scheduler is able to incorporate metaknowledge on the scenario level into the actual timeline planning at task level. In such a way, the initial pre-set priority assignment per task is changed according to the actual operational scenario.

4 A basic IA Task Scheduler

The presented object oriented task generation in combination with the intelligent agent radar scheduler is a basic initial design. Before extending this model however, we felt it to be necessary to design an actual scheduler, which proves the concept of filling a timeline based on cooperation between autonomous processes. This actual scheduler is termed 'Intelligent Agent Task Scheduler'.

In order to demonstrate the use of agents for planning tasks on a single resource, it suffices to use a blackboard model, which contains two timelines. Every task in the task set is instantiated by an agent, which only knows the particularities of its task (see 2.2). In addition to these task agents, an additional evaluator agent is introduced. The evaluator agent contributes by evaluating agent proposed timelines with regard to the yielded utility of that particular timeline.

Initially the Actual Time Line (ATL) is empty and the first agent is selected from the list of available agents. This list contains all agents that have not been able to

insert their task into the most actual timeline. Based on the task it represents, the agent attempts to place the task in an available space on the timeline. In doing so, the agent creates a so-called Proposed Time Line (PTL). If the agent is finished, the PTL is put on the blackboard next to the ATL. Placing a PTL on the blackboard triggers the Evaluator Agent to calculate the utility function according to form (1) for both ATL and PTL. If the PTL yields a higher utility than the ATL, the PTL is copied to the ATL, else the ATL remains as is. If the Evaluator accepts the PTL, then the proposing agent is removed from the agent list. In case an agent is not successful in finding a vacant position on the ATL, it can decide to remove an already planned agent from the ATL and insert its task instead. If the Evaluator accepts the PTL in that case, the removed agent is added to the list of agents again, so in turn it may try to get back into the ATL on another position. Fig. 6 depicts the agent blackboard environment as described above.

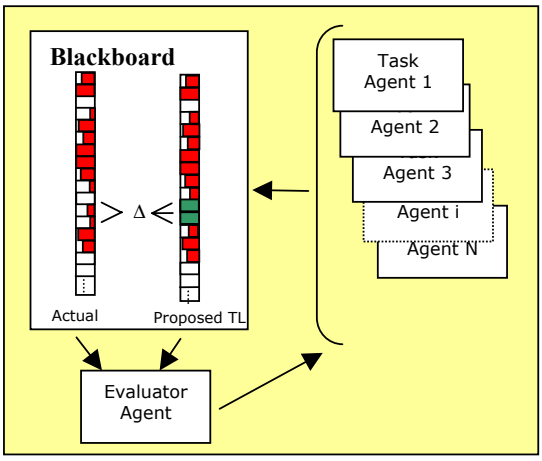

Fig. 6 A basic IA task scheduler

This simple mechanism only requires passive agents who wait in turn to propose a new timeline. There is no communication between the agents nor will the agents attempt to cooperate in order to have the overall utility maximized. As a result we have seen situations in which sub-optimal utility values have been achieved without the system being able to overcome this disadvantage. An easy initial attempt to reduce the risk of sub-optimization is to use several schedulers in parallel, each with a randomized sequence of agents on the agent list. By randomizing the order of the agents, every scheduler has a chance of either converging to another local optimum or even the overall optimum. By introducing such a parallelization, the advantage of approximating an NP-complete problem is severely reduced because of the need for increased processing capability for these parallel schedulers.

A more elegant way of reducing the risk of sub-optimization via local maxima is through increasing the level of cooperation between the agents. Cooperating agents need to be able to exchange information with other agents when a conflict between them occurs. Instead of letting every individual agent propose a modified timeline on a blackboard, the agents in conflict could evaluate more complex changes to the timeline, which involve an even larger group of agents. For further reading on coordination and agent negotiation, see [6].
Simulations of this simple IA task scheduler have shown that the scheduler is able to employ every planning algorithm as long as the task properties are well defined. Schedules based on alternative planning algorithms like First In First Out (FIFO), Shortest Processing time First (SPF), etc. are introduced via a simple modification to the task definition. Instead of assigning an à-priori value to the individual task priorities P_i, this value is derived from the planning algorithm. As an example, the P_i per task can be based on the task duration D_i via:

$$P_i = \frac{1}{D_i} * 100 \qquad (5)$$

Applying the same basic task scheduler to the tasks with this modified task definition has shown to lead to a SPF schedule. Similar approaches can be taken to create schedules based on alternative planning algorithms. Only by designing the proper priority allocation per task, the simple IA task scheduler is capable of incorporating a wide variety of existing planning algorithm

5 Extending the basic model

Before the Intelligent Agent Task Scheduler can be turned into an IA Radar Scheduler, several issues will have to be resolved. The following aspects of the design will be addressed in the course of 2004:

- Agent negotiation: The basic IA Task Scheduler employs a strategy by which an agent is allowed to remove another agent from the ATL. Like this, every agent action is limited to exchanging a task for another.

- Agent Learning: Although the need for learning is omitted from our simple example, more complex applications of task schedulers will have to incorporate adaptive strategies.

- Timeliness: As proven by the IA Task Scheduler, a MAS is capable of finding approximations of RSP. This approximation satisfies the requirements from para 2.4 with exception of timeliness. At this moment it is not clear whether the agent scheduler will be able to meet the time requirements imposed by the radar system.

Until now we defined a radar task as a single action with duration D to be accomplished in timeslot $[t_s, t_e]$. The agent tries to incorporate this task into a timeline without any flexibility in modifying the task definition. If it appears that this task cannot be included in the schedule, the task is removed from the task list and the originating object does not get the information to update the

attributes. Reality is fuzzier, in that sense that if the original task cannot be accepted, the agent might slightly modify the task definition in an attempt to get at least some part of the task executed. For example, a single track update at 50 msec could be split into two consecutive tasks, like for instance one at 25 msec and the next at 75 msec. The preferred option is the single update at the right time. If at that time the radar is saturated, preventing the planning of this task, executing the alternative task could provide the solution.

Agents should therefore be equipped with the capability to identify the various alternatives per task and to negotiate potential planning solutions in case of irresolvable conflicts. Identifying alternatives and selecting the most preferred as function of the planning situation belongs to the domain of agent learning. Resolving conflicts with a subset of agents involved in this situation is called agent negotiation. Both learning and negotiation cannot be studied independently in the case of radar task scheduling.

Negotiation between agents is extensively studied by Kraus et al. [6], culminating in the Strategic Negotiation Model. This model allows for an explicit and rational reasoning process by self-motivated or selfish agents without a central controller. In our IA Task Scheduler model we intend to implement an adapted Strategic Negotiation Model for agents to negotiate better Proposed Time Lines than by just replacing on or more agents by the proposing agent.

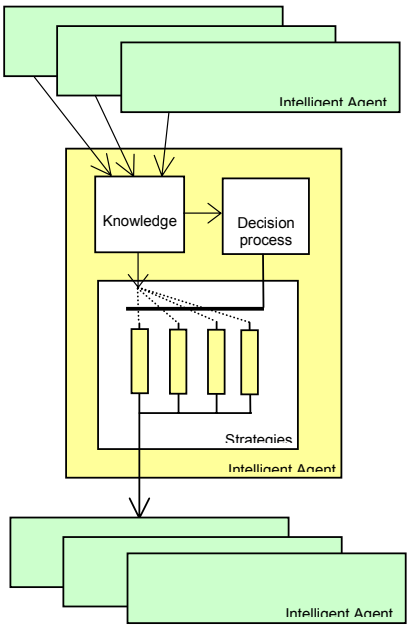

Fig. 7 A learning model for Task Agents

Fig. 7 depicts a simplified approach towards incorporating a learning mechanism into simple re-active agents. In this figure, information from other agents is fed into the agents' knowledgebase. This knowledgebase is explored in order to find supportive evidence for the selection of already defined courses of action. The agent selects such a strategy and continues his communication with other agents. This strategy selection process can be complemented with an à-posteriori evaluation of the success of the decision, in order to tune the selection process further.

Finally the issue of timeliness can be a factor that might not be easy to overcome. As demonstrated by examples of applications in the literature and by our own example of the Intelligent Agent Task Scheduler, the processing time overhead can increase substantially when the complexity of the scheduler increases. How this relationship between increasing complexity and required processing time is, needs to be addressed in future research.

6 Conclusions

The STATOR research project studies the possibilities to manage (radar) sensor based on the output of the radar to the operational user. Closing this feedback loop requires the incorporation of Operational and Technical Measures of Performance into the radar task scheduling process.

This process is NP-complete, which prohibits the use of an algorithm providing the optimum solution. Therefore, the radar scheduler needs to provide the best approximation to the problem, with 'best' being defined as complying with a wide variety of requirements. The OMOP and TMOP feedback loop emphasizes the need for an 'intelligent approach', which is found in the use of Intelligent Agent technology. A simple Intelligent Agent scheduler has proved to be able to provide a range of schedules depending on a user-selectable planning algorithm. Extending this simple model in the near future promises a capable and effective method of sensor tuning. If this method will be efficient enough to support real time radar scheduling needs to be determined.

References

[1] Dale R. Billeter, *Multi Function Array Radar*, Artech House, Norwood, MA, ISBN 0-89006-359-1, 1989.

[2] Michael R. Garey, David S. Johnson, *Computers and intractability, a Guide to the Theory of NP-Completeness*, W.H. Freeman and Company, New York, ISBN 0-7167-1044-7, 1979.

[3] Gerhard Weiss (ed.), *Multiagent Systems, A Modern Approach to Distributed Artificial Intelligence*, The MIT Press, Cambridge, MA, ISBN 0-262-23203-0, 1999.

[4] Weiming Shen, Douglas H. Norrie, Jean-Paul A. Barthès, *Multi-Agent Systems for Concurrent Intelligent Design and Manufacturing*, Taylor and Francis, New York, NY, ISBN 0-7484-0882-7, 2001.

[5] Mark d'Inverno, Michael Luck, Understanding Agent Systems, Springer-Verlag, Berlin, Germany, ISBN 3-540-41975-6, 1998.

[6] Sarit Kraus. *Strategic Negotiation in Multiagent Environments*. The MIT Press, Cambridge, MA, ISBN 0-262-11264-7, 2001.