An Agent-Based Meta-Level Architecture for Strategic Reasoning in Naval Planning

Mark Hoogendoorn[1], Catholijn M. Jonker[3], Peter-Paul van Maanen[1,2], and Jan Treur[1]

[1]Vrije Universiteit Amsterdam, Dept. of Artificial Intelligence De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands {mhoogen, pp, treur}@cs.vu.nl

[2]TNO Human Factors, Dept. of Information Processing, P.O. Box 23, 3769 ZG Soesterberg, The Netherlands

[3]Radboud University Nijmegen, Nijmegen Institute for Cognition and Information, Montessorilaan 3, 6525 HR Nijmegen, The Netherlands C.Jonker@nici.ru.nl

Abstract

The management of naval organizations aims at the maximization of mission success by means of monitoring, planning, and strategic reasoning. This paper presents an agent-based meta-level architecture for strategic reasoning in naval planning. The architecture is instantiated with decision knowledge acquired from naval domain experts and is formed into an executable agent-based model which is used to perform a number of simulation runs. To evaluate the simulation results, relevant properties for the planning decision are identified and formalized. These important properties are validated for the simulation traces.

1. Introduction

The management of naval organizations aims at the maximization of mission success by means of monitoring, planning, and strategic reasoning. In this domain, strategic reasoning more in particular helps in determining in resource-bounded situations if a go or no go should be given to, or to shift attention to, a certain evaluation of possible plans after an incident. An incident is an unexpected event, which results in an unmeant chain of events if left alone. Strategic reasoning in a planning context can occur both in *plan generation* strategies (cf. [15]) and *plan selection* strategies.

The above context gives rise to two important questions. Firstly, what possible plans are first to be considered? And secondly, what criteria are important for selecting a certain plan for execution? In resource-bounded situations first generated plans should have a high probability to result in a mission success, and the criteria to determine this should be as sound as possible.

In this paper a generic agent-based meta-level architecture (cf. [10]) is presented for planning, extended with a strategic reasoning level. Besides the introduction of an agent-based meta-level architecture, expert knowledge is used in this paper to formally specify executable properties for each of the components of the agent architecture. In contrast to other approaches, this can be done on a conceptual level. These properties can be used for simulation and facilitate formal validation by means of verification of the simulation results.

The agent architecture and its components are described in Section 2. Section 3 presents the method used to formalize the architecture. Section 4 presents each of the individual components on a more detailed level and instantiates them with knowledge from the naval domain. Section 5 describes a case study and discusses simulation results. In Section 6 a number of properties of the model's behavior are identified and formalized. A formal tool *TTL Checker* is used to check the validity of these properties in the simulated traces. Section 7 is a discussion.

2. An Agent-Based Meta-level Architecture for Naval Planning

The agent-based architecture has been specified using the DESIRE framework [2]. For a comparison of DESIRE with other agent-based modeling techniques, such as GAIA, ADEPT, and MetateM, see [13, 11]. The top-level of the system is shown in Figure 1 and consists of the ExternalWorld and the Agent. The ExternalWorld generates observations which are forwarded to the Agent, and executes the actions that have been determined by the

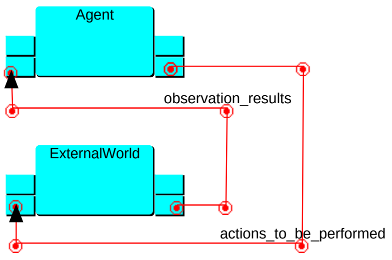

Fig. 1. Top-level architecture

Agent. The composition of the Agent is based on the generic agent model described in [3] of which two components are used: WorldInteractionManagement and OwnProcessControl, as shown in Figure 2. WorldInteractionManagement takes care of monitoring the observations that are received from the ExternalWorld. In case these observations are consistent with the current plan, the actions which are specified in the plan are executed by means of forwarding them to the top-level. Otherwise, evaluation information is generated and

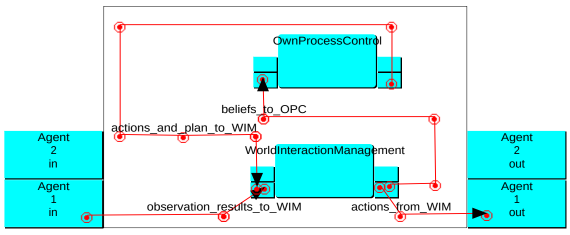

Fig. 2. Agent architecture

forwarded to the OwnProcessControl component. Once OwnProcessControl receives such an evaluation it determines whether the current plan needs to be changed, and in case it does, forwards this new plan to WorldInteractionManagement.

WorldInteractionManagement can be decomposed into two components, namely Monitoring and PlanExecution which take care of the tasks as previously presented (i.e. monitoring the observations and executing the plan). For the sake of brevity the Figure regarding these components has been omitted.

OwnProcessControl can also be decomposed, which is shown in Figure 3. Three components are present within OwnProcessControl: StrategyDetermination, PlanGeneration, and PlanSelection. The PlanGeneration component determines which plans are suitable, given the evaluation information received in the form of beliefs from WorldInteractionManagement, and the conditional rules given by StrategyDetermination. The candidate plans are forwarded to PlanSelection where the most appropriate plan is selected. In case no plan can be selected in PlanSelection this information is forwarded to the StrategyDetermination component. StrategyDetermination reasons on a meta-level (the input is located on a higher level as well as the output as shown in Figure 3), getting input by translating beliefs into reflected beliefs and by means of receiving the status

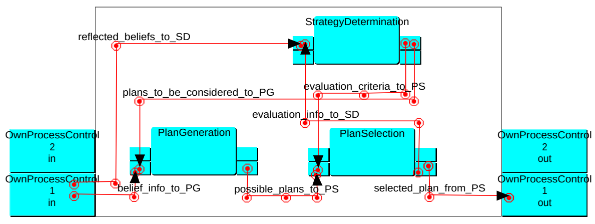

Fig. 3. Components within OwnProcessControl

of the plan selection process from PlanSelection. The component has the possibility to generate more conditional rules and pass them to PlanGeneration, or can change the evaluation criteria in PlanSelection by forwarding these criteria.

The model has some similarities with the model presented in [7]. A major difference is that an additional meta-level is present in the architecture presented here for the StrategyDetermination component. The advantage of having such an additional level is that the reasoning process will be more efficient, as the initial number of options are limited but are required to be the most straightforward ones.

3. Formalization Method

In this section the method used for the formalization of the model presented in section 2 is explained in more detail. To formally specify dynamic properties that are essential in naval strategic planning processes and therefore essential for the components within the agent, an expressive language is needed. To this end the Temporal Trace Language (TTL) is used as a tool; cf. [8]. In this section of the paper both an informal and formal representation of the properties are given.

A state ontology is a specification (in order-sorted logic) of a vocabulary. A state for ontology Ont is an assignment of truth-values {true, false} to the set At(Ont) of ground atoms expressed in terms of Ont. The *set of all possible states* for state ontology Ont is denoted by STATES(Ont). The set of *state properties* STATPROP(Ont) for state ontology Ont is the set of all propositions over ground atoms from At(Ont). A fixed *time frame* T is assumed which is linearly ordered. A *trace* or *trajectory* γ over a state ontology Ont and time frame T is a mapping $\gamma : T \rightarrow$ STATES(Ont), i.e., a sequence of states γ_t ($t \in T$) in STATES(Ont). The set of all traces over state ontology Ont is denoted by TRACES(Ont). Depending on the application, the time frame T may be dense (e.g., the real numbers), or discrete (e.g., the set of integers or natural numbers or a finite initial segment of the natural numbers), or any other form, as long as it has a linear ordering. The set of *dynamic properties* DYNPROP(Σ) is the set of temporal statements that can be formulated with respect to traces based on the state ontology Ont in the following manner.

Given a trace γ over state ontology Ont, the input state of a component c within the agent (e.g., PlanGeneration, or PlanSelection) at time point t is denoted by state(γ, t, input(c)). Analogously state(γ, t, output(c)) and state(γ, t, internal(c)) denote the output state, internal state and external world state.

These states can be related to state properties via the formally defined satisfaction relation |=, comparable to the Holds-predicate in the Situation Calculus: state(γ, t, output(c)) |= p denotes that state property p holds in trace γ at time t in the output state of agent-component c. Based on these statements, dynamic properties can be formulated in a formal manner in a sorted first-order predicate logic with sorts T for time points, Traces for traces and F for state formulae, using quantifiers over time and the usual first-order logical connectives such as $\neg, \wedge, \vee, \Rightarrow, \forall, \exists$. In trace descriptions, notations such as state(γ, t, output(c))|= p are shortened to output(c)|p.

To model direct temporal dependencies between two state properties, the simpler *leads to* format is used. This is an executable format defined as follows. Let α and β be state properties of the form 'conjunction of literals' (where a literal is an atom or the negation of an atom), and e, f, g, h non-negative real numbers. In the *leads to* language $\alpha \twoheadrightarrow_{e, f, g, h} \beta$, means:

if state property α holds for a certain time interval with duration g, then after some delay (between e and f) state property β will hold for a certain time interval of length h.

For a precise definition of the *leads to* format in terms of the language TTL, see [9]. A specification of dynamic properties in *leads to* format has as advantages that it is executable and that it can easily be depicted graphically.

4. Component Specification for Naval Planning

This Section introduces each of the components within the strategic planning process in more detail. The components presented in this section are only those part of OwnProcessControl within the agent as they are most relevant for the planning process. A partial specification of executable properties in semi-formal format is also presented for each of these components. The properties introduced in this Section are generic for naval (re)planning and can easily be instantiated with mission specific knowledge. All of these properties are the result of interviews with officers of the Royal Netherlands Navy.

4.1 Plan Generation

The rules for generation of a plan can be stated very generally as the knowledge about plans. Conditions for those plans are stored in the StrategyDetermination component, which is treated later. Basically, in this domain the component contains one rule:

```
if      belief(S:SITUATION, pos)
 and    conditionally_allowed(S:SITUATION, P:PLAN)
then    candidate_plan(P:PLAN)
```

Stating that in case Monitoring evaluated the current situation as being situation S and the PlanGeneration has received an input that situation S allows for plan P then it is a candidate plan. This information is passed to the PlanSelection component.

4.2 Plan Selection

Plan selection is the next step in the process and for this domain there are three important criteria that determine whether a plan is appropriate or not: (1) Mission success; (2) safety, and (3) fleet morale criterion. In this scenario it is assumed that a weighed sum can be calculated and used in order to make a decision between candidate plans. The exact weight of each criterion is determined by the StrategyDetermination component. The value for the criteria can be derived from observations in the world and for example a weighed sum can be taken over time. To obtain the observations, for each candidate plan the consequence events of the plan are determined and formed into an observation. Thereafter the consequences of these observations for the criteria can be determined. In the examples shown below the bridge between changes of the criteria after an observation and the overall value of the criteria are not shown in a formal form for the sake of brevity.

Mission Success

An important criterion is of course the mission success. Within this criterion the objective of the mission plays a central role. In case a certain decision needs to be made, the influence this decision has for the mission success needs to be determined. The criterion involves taking into account several factors. First of all, the probability that the deadline is reachable. Besides that, the probability that the mission succeeds with a specific fleet configuration. The value of the mission success probability is a real number between 0 and 1. A naval domain expert has labeled certain events with an impact value on mission success. This can entail a positive effect or a negative effect. The mission starts with an initial value for success, taking into consideration the assignment and the enemy. In case the situation changes this can lead to a change of the success value. An example of an observation with a negative influence is shown below.

```
if      current_success_value(S:REAL)
 and    belief(ship_left_behind, pos)
then    new_succes_value(S:REAL * 0.8)
```

Safety

Safety is an important criterion as well. When a ship loses propulsion the probability of survival decreases dramatically if left alone. Basically, the probability of survival depends on three factors: (1) the speed with which the task group is sailing; (2) the configuration of own ships, which includes the amount and type of ships, and their relative positions; (3) the threat caused by the enemy, the kind of ships the enemy has, the probability of them attacking the task group, etc.

The safety value influences the evaluation value of possible plans. The duration of a certain safety value determines its weight in the average risk value, so a weighed sum based on time duration is taken. The value during a certain period in time is again derived by means of an initial safety value and events in the external world causing the safety value to increase or decrease. An example rule:

```
if      current_safety_value(S:REAL)
 and    belief(speed_change_from_to(full, slow), pos)
then    new_safety_value(0.5 * S:REAL)
```

Fleet morale

The morale of the men on board of the ships is also important as criterion. Morale is important in the considerations as troops with a good morale are much more likely to win compared to those who do not have a good morale. Troop morale is represented by a real number with a value between 0 and 1 and is determined by events in the world observed by the men. Basically, the men start with a certain morale value and observations of events in the world can cause the level to go up or down, similar to the mission success criterion. One of the negative experiences for morale is the observation of being left behind without protection or seeing others solely left behind:

```
if      current_morale_value(M:REAL)
 and    belief(ship_left_behind, pos)
then    new_morale_value(M:REAL * 0.2)
```

An observation increasing the morale is that of sinking an enemy ship:

```
if      current_morale_value(M:REAL)
and     belief(enemy_ship_eliminated, pos)
and     min(1, M:REAL * 1.6, MIN:REAL)
then    new_morale_value(MIN:REAL)
```

4.3 Strategy Determination

The StrategyDetermination component within the model has two functions: First of all, it determines the conditional plans that are to be used given the current state. Secondly, it provides a strategy for the selection of these plans.

In general, naval plans are generated according to a preferred plan library or in exceptional cases outside of this preferred plan library. The StrategyDetermination component within the model determines which plans are to be used and thereafter forwards these plans to the PlanGeneration component. The StrategyDetermination component determines one of three modes of operation on which conditional rules are to be used in this situation:

1. **Limited action demand.** This mode is used as an initial setting and is a subset of the preferred plan library. It includes the more common actions within the preferred plan library;
2. **Full preferred plan library.** Generate all conditional rules that are allowed according to the preferred plan library. This mode is taken when the limited action mode did not provide a satisfactory solution;
3. **Exceptional action demand**. This strategy is used in exceptional cases, and only in case the two other modes did not result in an appropriate candidate plan.

Next to determining which plans should be evaluated, the StrategyDetermination component also determines *how* these plans should be evaluated. In Section 4.3 it was stated that the plan selection depends on mission success, safety, and fleet morale. All three factors determine the overall evaluation of a plan to a certain degree. Plans can be evaluated by means of an evaluation formula, which is described by a weighted sum. Differences in weights determine differences in plan evaluation strategy. The plan evaluation formula is as follows (in short):

evaluation_value(P:PLAN) = α * mission_success_value(P:PLAN) + β * safety_value(P:PLAN) + γ * fleet_morale_value(P:PLAN)

where all values and degrees are in the interval [0,1], and $\alpha + \beta + \gamma = 1$. The degrees depend on the type of mission and the current state of the process. For instance, if a mission is supposed to be executed safely at all cost or the situation shows that already many ships have been lost, the degree β should be relatively high.

In this case the following rules hold:

```
if      problem_type(mission_success_important)
 and    problem_type(safety_important)
 and    problem_type(fleet_morale_important)
 and    candidate_plan(P:PLAN)
 and    mission_success_value(P:PLAN, R1:REAL)
 and    safety_value(P:PLAN, R2:REAL)
 and     fleet_morale_value(P:PLAN, R3:REAL)
then    evaluation_value(no_propulsion(ship),
                    0.33 * R1:REAL + 0.33 * R2:REAL + 0.33 *R3:REAL)
```

In case two criteria are most important the following rule holds:

```
if      problem_type(mission_success_important)
 and    problem_type(safety_important)
 and    not problem_type(fleet_morale_important)
 and    candidate_plan(P:PLAN)
 and    mission_success_value(P:PLAN, R1:REAL)
 and    safety_value(P:PLAN, R2:REAL)
 and     fleet_morale_value(P:PLAN, R3:REAL)
then    evaluation_value(no_propulsion(ship),
                     0.45 * R1:REAL + 0.45 * R2:REAL + 0.1 *R3:REAL)
```

This holds for each of the problem type combinations where two criteria are important: A weight of 0.45 in case the criterion is important for the problem type and 0.1 otherwise. Finally, only one criterion can be important:

```
if      problem_type(mission_success_important)
 and    not problem_type(safety_important)
 and    not problem_type(fleet_morale_important)
 and    candidate_plan(P:PLAN)
 and    mission_success_value(P:PLAN, R1:REAL)
 and    safety_value(P:PLAN, R2:REAL)
 and    fleet_morale_value(P:PLAN, R3:REAL)
then    evaluation_value(no_propulsion(ship),
                           0.6 * R1:REAL + 0.2 * R2:REAL + 0.2 *R3:REAL)
```

The plan generation modes and plan selection degrees presented above can be specified by formal rules which have been omitted for the sake of brevity.

5. Case-study: Total Steam Failure

This Section presents a case study which has been formalized using the agent-based model presented in Section 2 and 4. This case study is again based upon interviews with expert navy officers of the Royal Netherlands Navy. The formalization of this process follows the methodology presented in Section 3.

5.1 Scenario Description

The scenario used as an example is the first phase within a *total steam failure* scenario. A fleet consisting of 6 frigates (denoted by F1 – F6) and 6 helicopters (denoted by H1 – H6) are protecting a specific area called Zulu Zulu (denoted by ZZ). For optimal protection of valuable assets that need to be transported to a certain location, and need to arrive before a certain deadline, the ships carrying these assets are located in ZZ. These ships should always maintain their position in ZZ to guarantee optimal protection. The formation at time T0 is shown in Figure 4. On that same time-point the following incident occurs: An amphibious transport ship, that is part of ZZ, loses its propulsion and cannot start the engines within a few minutes. When a mission is assigned to a commander of the task group (CTG), he receives a preferred plan library from the higher echelon. This library gives an exhaustive list of situations and plans that are allowed to be executed within that situation. Therefore the CTG has to make a decision: What to do with the ship and the rest of the fleet. In the situation occurring in the example scenario the preferred plan library consists of four plans:

1. **Continue sailing.** Leave the ship behind. The safety of the main fleet will therefore be maximal, however the risk for the ship is high. The morale of all the men within the fleet will drop.

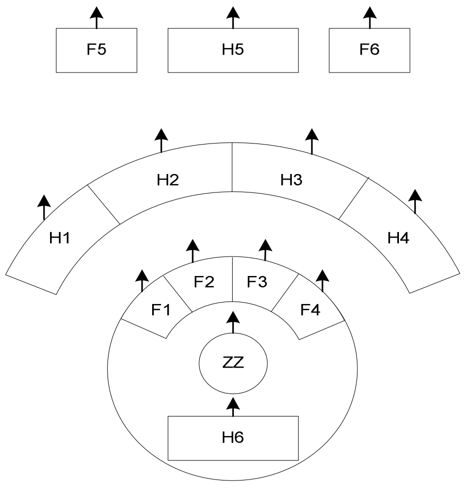

Fig. 4. Scenario for meta-reasoning

2. **Stop the entire fleet.** Stopping the fleet ensures that the ship is not left behind and lost, however the risks for the other ships increase rapidly as an attack is more likely to be successful when not moving.
3. **Return home without the ship.** Rescue the majority of the men from the ship, return home, but leave a minimal crew on the ship that will still be able to fix the ship. The ship will remain in danger until it is repaired and the mission is surely not going to succeed. The morale of the men will drop to a minimal level. This option is purely hypothetical according to the experts.
4. **Form a screen around the ship.** This option means that part of the screen of the main fleet is allocated to form a screen around the ship. Therefore the ship is protected and the risks for the rest of the fleet stay acceptable.

Option 4 involves a lot more organizational change compared to the other options and is therefore considered after the first three options. The CTG decides to form a screen around the ship.

5.2 Simulation Results

The most interesting results of the simulation using the architecture and properties described in Section 2 and 4, and instantiated with the case-study specific knowledge from Section 5.1 are shown in Figure 5. The trace, a temporal description of chains of events, describes the decision making process of the agent which pays the role of Commander Task Group (CTG). The atoms on the left side denote the information between and within the components of the agents. To keep the Figure clear only the atoms of the components on the lowest level of the agent architecture are shown. The right side of the figure shows when these atoms are true. In case of a black box the atom is true during that period, in the other cases the

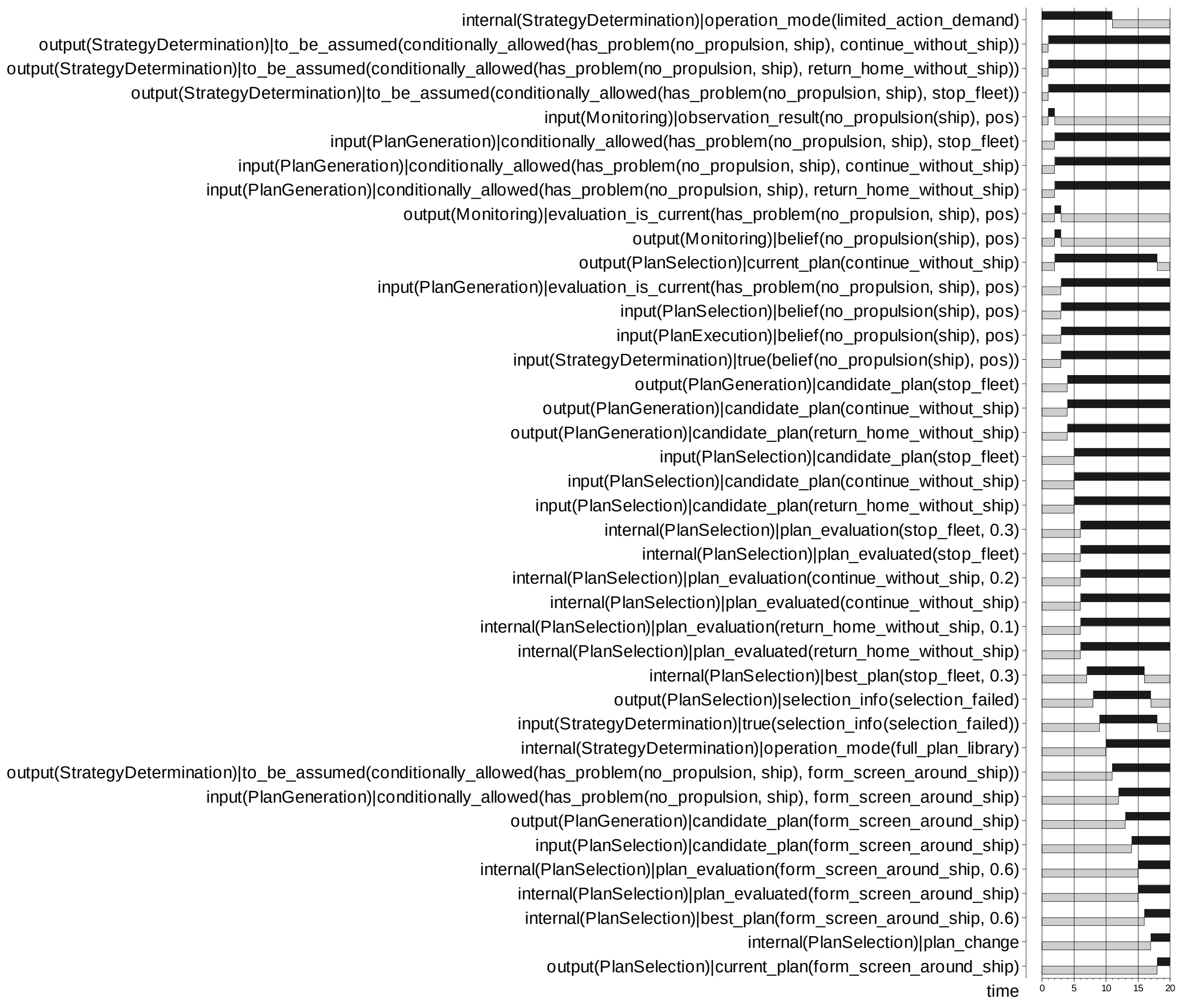

Fig. 5. Trace of the total steam failure simulation

atom is false (closed world assumption). The atoms used are according to the model presented in Section 2. For example, internal(PlanGeneration) denotes that the atom is internal within the PlanGeneration component. More specifically, the trace shows that at time-point 1 the Monitoring component receives an input that the ship has no propulsion

input(Monitoring)|observation_result(no_propulsion(ship), pos)

The current plan is to continue without the ship, as the fleet continues to sail without any further instructions:

output(PlanSelection)|current_plan(continue_without_ship)

As the StrategyDetermination component always outputs the options currently available for all sorts of situations (in this case only a problem with the propulsion of a ship) it continuously outputs the conditionally allowed information in the limited action mode, for example:

output(StrategyDetermination)|to_be_assumed(
conditionally_allowed(has_problem(no_propulsion,
ship),continue_without_ship))

The information becomes an input through downward reflection, a translation from a meta-level to a lower meta-level: input(PlanGeneration)|conditionally_allowed(
has_problem(no_propulsion, ship), continue_without_ship)

The Monitoring component forwards the information about the observation to the components on the same level as beliefs. The StrategyDetermination component also receives this information but instead of a belief it arrives as a reflected belief through upward reflection which is a translation of information at a meta-level to a higher meta-level:

input(StrategyDetermination)|
true(belief(no_propulsion(ship), pos))

Besides deriving the beliefs on the observations the Monitoring component also evaluates the situation and passes this as evaluation info to the PlanGenerator.

input(PlanGenerator)|evaluation(has_problem(no_propulsion,
ship), pos)

This information acts as a basis for the PlanGenerator to generate candidate plans, which are sent to the PlanSelection, for example.

input(PlanSelection)|candidate_plan(continue_without_ship)

Internally the PlanSelection component determines the evaluation value of the different plans, compares them and derives the best plan out of the candidate plans:

internal(PlanSelection)|best_plan(stop_fleet, 0.3)

This value is below the threshold evaluation value and therefore the PlanSelection component informs the StrategyDetermination component that no plan has been selected:

output(PlanSelection)|selection_info(selection_failed)

Thereafter the StrategyDetermination component switches to the full preferred plan library and informs PlanGeneration of the new options. PlanGeneration again generates all possible plans and forwards them to PlanSelection. PlanSelection now finds a plan that is evaluated above the threshold and makes that the new current plan.

output(PlanSelection)|current_plan(form_screen_around_ship)

This plan is forwarded to the PlanExecution and Monitoring components (not shown in the trace) and is executed and monitored.

6. Validation by Verification

After that a formalized trace has been obtained in the previous section, either by formalization of an empirical trace or by means of simulation, in this section it is validated whether the application of the model complies to certain desired properties of this trace. Below the verification of these properties in the trace are shown. The properties are independent from the specific scenario and should hold for every scenario for which the agent-based meta-level architecture presented in Section 2 and 4 is applied. The properties are formalized using Temporal Trace Language as described in Section 3.

P1: Upward reflection

This property states that information generated at the level of the Monitoring and PlanSelection components should always be reflected upwards to the level of the StrategyDetermination component. In semi-formal notation:

At any point in time t,
if Monitoring outputs a belief about the world at time t
then at a later point in time t2 StrategyDetermination receives this information through upward reflection
At any point in time t,
if PlanSelection outputs selection info at time t
then at a later point in time t2 StrategyDetermination receives this information though upward reflection.

In formal form the property is as follows:

$\forall$t [[$\forall$O:OBS, S:SIGN
[state(γ, t, output(Monitoring)) |= belief(O, S)
$\Rightarrow$ $\exists$t2 $\geq$ t state(γ, t2, input(StrategyDetermination)) |= true(belief(O,S))]]
& [$\forall$SI:SEL_INFO [state(γ, t, output(PlanSelection)) |= selection_info(SI)
$\Rightarrow$ $\exists$t2 $\geq$ t state(γ, t2, input(StrategyDetermination)) |= true(selection_info(SI))]]]

This property has been automatically checked and thus shown to be satisfied within the trace.

P2: Downward reflection

Property P2 verifies that all information generated by the StrategyDetermination component for a lower meta-level is made available at that level through downward reflection. In formal form:

$\forall$t, S:SITUATION, P:PLAN [state(γ, t, output(StrategyDetermination)) |= to_be_assumed(conditionally_allowed(S, P))
$\Rightarrow$ $\exists$t2 $\geq$ t state(γ, t2, input(PlanGeneration)) |= conditionally_allowed(S, P)]

This property is also satisfied for the given trace.

P3: Extreme measures

This property states that measures that are not part of the preferred plan library (extreme measures) are only taken in case some other options failed. In formal form:

$\forall$t, t2 > t, S:SITUATION, P1:PLAN, P2:PLAN
[[state(γ, t, output(Monitoring)) |= evaluation(exception(S), pos) &
state(γ, t, output(PlanSelection)) |= current_plan(P1) &
state(γ, t2, output(PlanSelection)) |= current_plan(P2) & P1 $\neq$ P2
& $\neg$state(γ, t2, internal(StrategyDetermination)) |= to_be_assumed(preferred_plan(S, P2)]
$\Rightarrow$ $\exists$t' [t' $\geq$ t & t' $\leq$ t2 & state(γ, t', output(PlanSelection)) |= selection_info(selection_failed)]]

The property is satisfied for the given trace.

P4: Plans are changed only if an exception was encountered

Property P4 formally describes that a plan is only changed in case there has been an exception that triggered this change. Formal:

$\forall$t, t2 $\geq$ t, P:PLAN [[state(γ, t, output(PlanSelection)) |= current_plan(P) &
$\neg$state(γ, t2, output(PlanSelection)) |= current_plan(P)]
$\Rightarrow$ $\exists$t', S:SITUATION [t' $\geq$ t & t' $\leq$ t2 & state(γ, t', output(Monitoring)) |= evaluation(exception(S), pos)]]

This property is again satisfied for the given trace.

7. Discussion

This paper presents an agent-based architecture for strategic planning (cf. [15]) for naval domains. The architecture was designed as a meta-level architecture (cf. [10]) with three levels. The interaction between the levels in this paper is modeled by reflection principles (e.g., [1]). The dynamics of the architecture is based on a multi-level trace approach as an extension of what is described in [6]. The architecture has been instantiated with naval strategic planning knowledge. The resulting executable model has been used to perform a number of simulation runs. To evaluate the simulation results desired properties for the planning decision process have been identified, formalized, and then validated for the simulation traces.

A meta-level architecture for strategic reasoning in another area, namely that of design processes is described in [4]. This architecture has been used as a source of inspiration for the current architecture for strategic planning. In other architectures, such as in PRS [5], meta-level knowledge is also part of the system, however this knowledge is not explicitly part of the architecture (it is part of the Knowledge Areas) as is the case in the architecture presented in this paper.

Agent models of military decision making have been investigated before. In [14] for example an agent based model is presented that mimics the decision process of an experienced military decision maker. Potential decisions are evaluated by checking if they are good for the current goals. A case study of decisions to be made at an amphibian landing mission is used. The outcome of the evaluations of the decisions that can be made in the case-study are compared to the decisions made by real military commanders. The approach presented is different from the approach taken in this paper as a more formal approach is taken here to evaluate the model created. Also the focus in this paper is more on the model of the decision maker itself and not on the correctness of the decisions, which is the case in [14]. The main advantage of the approach taken is that the system is specified and can be simulated on a conceptual level contrary to other approaches. Finally, this paper addressed resource-bounded situations. In [12] an overview is presented of models for human behavior that can be used for simulations. Similar to research done in other agent-based systems using the DESIRE framework [2], future research in simulation and the validation of relevant properties for the resulting simulation traces is expected to give key insight for the implementation of future complex resource-bounded agent-based planning support systems used by commanders on naval platforms.

Acknowledgements

CAMS-Force Vision, a software development company associated with the Royal Netherlands Navy, funded this research and provided domain knowledge. The authors especially want to thank Jaap de Boer (CAMS-Force Vision) for his expert knowledge.

References

1. Bowen, K. and Kowalski, R., Amalgamating language and meta-language in logic programming. In: K. Clark, S. Tarnlund (eds.), Logic programming. Academic Press, 1982.
2. Brazier, F.M.T., Dunin Keplicz, B., Jennings, N., and Treur, J., DESIRE: Modelling Multi-Agent Systems in a Compositional Formal Framework. *International Journal of Cooperative Information Systems*, vol. 6, 1997, pp. 67-94.
3. Brazier, F.M.T., Jonker, C.M., and Treur, J., Compositional Design and Reuse of a Generic Agent Model. *Applied Artificial Intelligence Journal*, vol. 14, 2000, pp. 491-538.
4. Brazier, F.M.T., Langen, P.H.G. van, and Treur, J., Strategic Knowledge in Design: a Compositional Approach. Knowledge-based Systems, vol. 11, 1998 (Special Issue on Strategic Knowledge and Concept Formation, K. Hori, ed.), pp. 405-416.
5. Georgeff, M. P., and Ingrand, F. F., Decision-making in an embedded reasoning system. In Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89), pages 972-978, Detroit, MI, 1989.
6. Hoek, W. van der, Meyer, J.-J.Ch., and Treur, J., Formal Semantics of Meta-Level Architectures: Temporal Epistemic Reflection. International Journal of Intelligent Systems, vol. 18, 2003, pp. 1293-1318.
7. Jonker, C.M., and Treur, J., A Compositional Process Control Model and its Application to Biochemical Processes. Applied Artificial Intelligence Journal, vol. 16, 2002, pp. 51-71.
8. Jonker, C.M., and Treur, J. Compositional verification of multi-agent systems: a formal analysis of pro-activeness and reactiveness. International. Journal of Cooperative Information Systems, vol. 11, 2002, pp. 51-92.
9. Jonker, C.M., Treur, J., and Wijngaards, W.C.A., A Temporal Modelling Environment for Internally Grounded Beliefs, Desires and Intentions. Cognitive Systems Research Journal, vol. 4, 2003, pp. 191-210.
10. Maes, P, Nardi, D. (eds), Meta-level architectures and reflection, Elsevier Science Publishers, 1988.
11. Mulder, M, Treur, J., and Fisher, M., Agent Modelling in MetateM and DESIRE. In: M.P. Singh, A.S. Rao, M.J. Wooldridge (eds.), *Intelligent Agents IV, Proc. Fourth International Workshop on Agent Theories, Architectures and Languages, ATAL'97.* Lecture Notes in AI, vol. 1365, Springer Verlag, 1998, pp. 193-207.
12. Pew, R.W. and Mavor, A.S.. Modeling Human and Organizational Behavior, National Academy Press, Washington, D.C. 1999.
13. Shehory, O., and Sturm, A., Evaluation of modeling techniques for agent-based systems, In: Proceedings of the fifth international conference on Autonomous agents, Montreal, Canada, May 2001, pp. 624-631.
14. Sokolowski, J., Enhanced Military Decision Modeling Using a MultiAgent System Approach, In Proceedings of the Twelfth Conference on Behavior Representation in Modeling and Simulation, Scottsdale, AZ., May 12-15, 2003, pp. 179-186.
15. Wilkins, D.E., Domain-independent planning Representation and plan generation. Artificial Intelligence 22 (1984), pp. 269-301.