# Agent-Based Analysis and Simulation of Meta-Reasoning Processes in Strategic 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 1081, 1081 HV Amsterdam, The Netherlands
{mhoogen, treur}@cs.vu.nl
[2] TNO Human Factors, Dept. of Human in Command,
P.O. Box 23, 3769 ZG Soesterberg, The Netherlands
peter-paul.vanmaanen@tno.nl
[3] Delft University of Technology, Dept. of Man-Machine Interaction,
Mekelweg 4, 2628 CD Delft, The Netherlands
C.M.Jonker@tudelft.nl

**Abstract.** This paper presents analysis and simulation of meta-reasoning processes based on an agent-based meta-level architecture for strategic reasoning in naval planning. The architecture was designed as a generic agent model and instantiated with decision knowledge acquired from naval domain experts and was specified as an executable agent-based model which has been used to perform a number of simulations. To evaluate the simulation results, relevant properties for the planning decision were identified and formalized. These properties have been validated for the simulation traces.

**Keywords:** Meta-reasoning, simulation, planning, intelligent agent systems.

## 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, just as well as in all other domains that are characterized by their resource-boundedness, plan generation and action selection are supported by strategic reasoning; for example, it may help to determine to decide whether a go or no go should be given to a certain possible plan after an incident, whether this should be investigated further, or even whether attention should be shifted to a different plans altogether. 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. [21]) and plan selection strategies (cf. [10, 20]).

The above context gives rise to two important questions. Firstly, what possible (candidate) plans are there to be considered? And secondly, what criteria should one use in order to come to a set of possible candidate plans and what criteria should one use to select a certain plan from such sets for execution? In resource-bounded

situations the plan generation process should be limited, i.e., the first generated plans should have a high probability to result in a mission success, and the criteria for selection should be as sound as possible. Furthermore, among the specific evaluation criteria for plans that are appropriate in the naval domain are aspects such as mission success, troop morale, and safety of the ships and troops. All these aspects are introduced and formalized in this paper.

In the literature on meta-reasoning and meta-level architectures, one of the selling points mentioned often is that such a system is able to reflect on its own knowledge state and reasoning process, and therefore can introduce reasonable additional assumptions and make more efficient choices for the reasoning process; e.g., [3, 7, 9. 13, 18, 19]. Real applications, based on these promises, developed in cooperation with domain experts, however, are rarely found in the literature.

In this paper a generic agent-based meta-level architecture (cf. [13]) is presented for planning, extended with a strategic reasoning level. The architecture has been designed and formally specified using the component-based design method for agent systems DESIRE; cf. [4]. In cooperation with domain experts, it is shown how this formal specification has been applied in the naval domain. For this application expert knowledge is used to identify and formally specify executable dynamic properties for each of the components within the generic agent architecture. This has been done on a conceptual level. The executable properties have been specified in the executable temporal language LEADSTO (Language and Environment for Analysis of Dynamics by SimulaTiOn [1]) and were used for simulation within the LEADSTO software environment. Moreover, mainly based on the input of the domain experts, more global dynamic properties for larger parts of the reasoning process have been identified and formally specified in the expressive temporal language TTL (Temporal Trace Language [2]). The latter properties have been formally verified against the simulation traces automatically, using the TTL Checker software environment; thus validation of the executable model was obtained.

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. The tool TTL Checker is used to automatically 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 component-based design method for agent systems DESIRE [4]. For a comparison of DESIRE with other agent-based modeling techniques, such as GAIA, ADEPT, and MetateM, see [14, 16]. Note that this architecture concerns a multi-agent system, this paper however only describes the architecture of a single agent. 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 Agent. The composition of the Agent is based on the generic

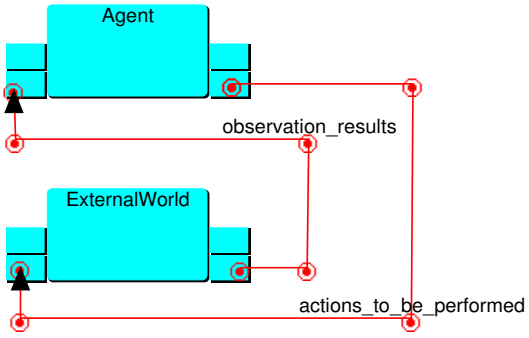


**Fig. 1.** Top-level architecture

agent model described in [6] 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 forwarded to the OwnProcessControl component. Once OwnProcessControl receives such an evaluation it

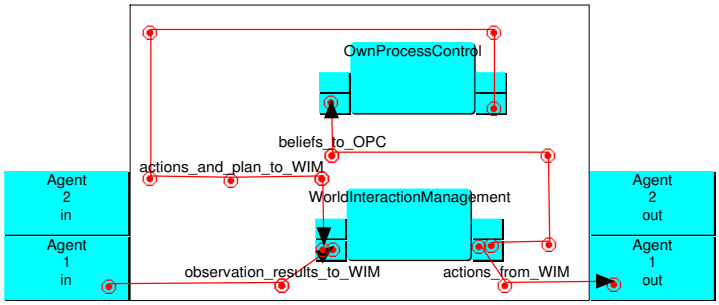


**Fig. 2.** Agent architecture

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

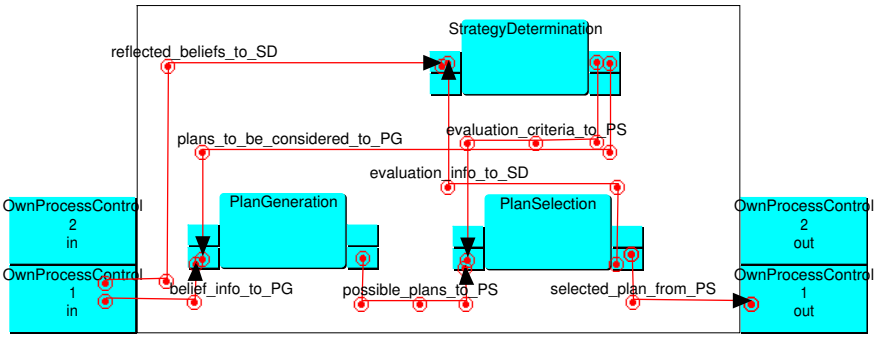


**Fig. 3.** Components within OwnProcessControl

status 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 [11]. 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. [12]. 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* $\gamma$ over a state ontology Ont and time frame T is a mapping $\gamma : T \rightarrow STATES(Ont)$, i.e., a sequence of states $\gamma_t$ ($t \in T$) 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 ¬, ∧, ∨, ⇒, ∀, ∃. 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 $\alpha \rightarrow\!\!\!\!\rightarrow_{e, f, g, h} \beta$, 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 [12]. 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 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.

Note that the plans within the first mode of operation occur much more frequently than the ones in the second mode, a similar relation holds between the second and the third mode of operation. As a result of this frequency difference, having such a strategy determination component improves the efficiency of the reasoning process. 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 $\beta$ should be relatively high.

In case of equally important criteria the following rule holds:

```
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-studies

This Section presents several case studies which have been formalized using the agent-based model presented in Section 2 and 4. These case studies are 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. Three case studies are presented: total steam failure, submarine threat, and frigate loss.

### 5.1 Total Steam Failure

The first scenario used as a case study is called total steam failure. First, the scenario is described, after which the simulation results are presented.

#### 5.1.1 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

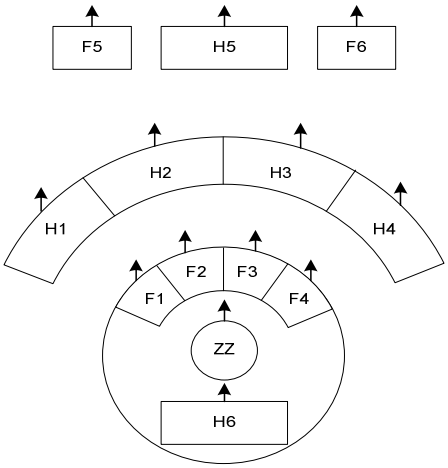


**Fig. 4.** Scenario for meta-reasoning

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 at a maximum level, however the risk for the ship is high. The morale of all the men within the fleet will drop.
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

### 5.1.2 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 plays the role of Commander Task Group (CTG). The atoms on the left side denote the information between and within the components of the agent. To keep the Figure clear only the atoms of the components on the lowest level of the agent architecture are shown. The

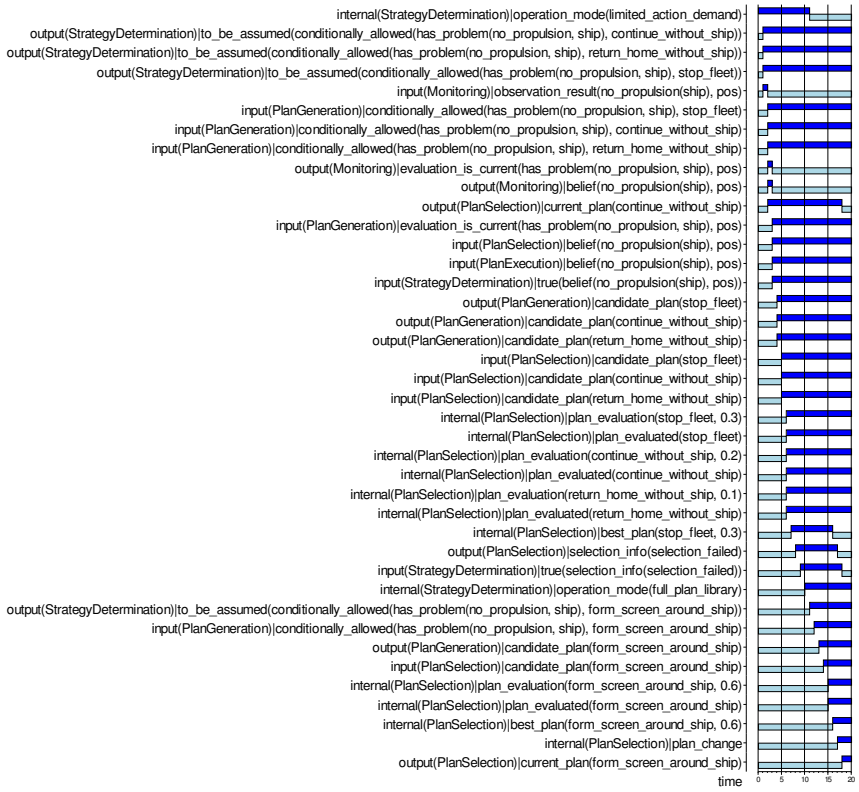


**Fig. 5.** Trace of the total steam failure simulation

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 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.

### 5.2 Submarine Threat

The second scenario is called submarine threat, and deals with a hostile submarine being detected within the fleet. First, a description of the scenario is given and thereafter simulation results are presented.

#### 5.2.1 Scenario Description

The initial fleet formation and mission for this scenario is identical to the one explained in Section 5.1.1. Another event however occurs that needs to be dealt with. Frigate F1 suddenly detects sonar contact with a high probability that it concerns a hostile submarine. The position of this submarine is such that the assets in Zulu Zulu are within torpedo range of the submarine. The plan library for the CTG in this particular situation is as follows:

1. **Eliminate and turn.** This option consists of two actions: First of all, F1 will fire a torpedo in the direction of the detected submarine. Thereafter, several frigates are sent to eliminate the submarine whereas the remainder of the fleet turns away from the submarine, positioning several frigates between the submarine and Zulu Zulu. This option results in risk for the frigates chasing the submarine whereas the remainder of the fleet remains relatively safe. Morale of the men will go up, and mission success is not so much endangered.
2. **Full attack.** This plan entails a full attack on the submarine with all available resources. Disadvantage is however that Zulu Zulu is no longer protected, and another enemy ship could possibly attack Zulu Zulu. The risk for mission success is therefore high, and morale of the men on board of the ships part of Zulu Zulu will drop, since they are being left behind without protection.
3. **Full throttle.** Accelerate to maximum speed, in order to try and outrun the submarine, zig zag to avoid the submarine getting a lock on one of the ships within Zulu Zulu. Morale of the troops will go down since they know there is a submarine somewhere trying to attack, and mission success will be much lower as well since the submarine might have the ability to successfully fire torpedos at Zulu Zulu. Safety is also low.

Option 3 is considered only after the first two have been considered as trying to escape from a submarine is highly dangerous and therefore seriously threatens mission success. Preferred plan is therefore to try and eliminate the submarine. The CTG decides to choose the eliminate and turn plan.

### 5.2.2 Simulation Results


**Fig. 6.** Trace of the submarine threat simulation

Figure 6 shows the results of a simulation of the submarine threat scenario. Initially, again the operation mode is set to limited action demand, which results in two plans being outputted by the StrategyDetermination component:

output(StrategyDetermination)|to_be_assumed(conditionally_allowed(has_problem(submarine_detected), ship), eliminated_and_turn)
output(StrategyDetermination)|to_be_assumed(conditionally_allowed(has_problem(submarine_detected), ship), full_attack)

Suddenly, an event occurs which is precisely the event for which these conditional plans are meant, namely that a submarine has been detected by a ship:

output(Monitoring)|belief(detected(submarine), pos)

As a result the current plan selected to handle the situation is again to continue with the current plan, which is to continue sailing. The PlanGeneration component generates the currently available plans for handling the event, which it has received from the StrategyDetermination component:

output(PlanGeneration)|cadidate_plan(eliminated_and_turn)
output(PlanGeneration)|cadidate_plan(full_attack)

This output is received by the PlanSelection component, which starts to evaluate the two available plans. After evaluation, the plan to eliminate and turn is found to be best

and is evaluated above the threshold value. As a result, it is selected as the new current plan:

```
output(PlanSelection)|current_plan(elminate_and_turn)
```

As can be seen in the simulation, only two out of three available plans have been evaluated before selecting a new plan. Since the plans being evaluated first are the ones typically best suitable in the situation, this saves a lot of precious evaluation most of the time.

### 5.3 Frigate Loss

Final scenario which has been investigated is that of a frigate being hit by a submarine torpedo.

#### 5.3.1 Scenario Description

Again, the initial fleet configuration and mission are identical to the description presented in Section 5.1.1. Again, a submarine is detected, for which the CTG decides to send in H3 to eliminate the submarine. The submarine however fires a torpedo which strikes F3 causing it to sink. There are now several options how to continue:

1. **Eliminate and save.** Eliminate the submarine first by reinforcing the current attack units. Thereafter, save the drowning crew of frigate F3. This option maximizes the morale of the troops as they see their colleagues being saved, mission success is however slightly endangered as picking up the drowning crew will result in frigates lying still, which makes them more vulnerable for enemy attacks.
2. **Save crew.** Immediately use all resources to save the crew on board of the sunken ship. In this scenario this is devastating for mission success as the submarine can easily attack the ships within Zulu Zulu. Furthermore, the submarine could even attack the resources that are being used to save the crew of the sunken ship. The safety for the crew of the sunken ship is relatively high whereas the safety for the other ships is low.
3. **Surrender.** Hoist the white flag and surrender to avoid further casualties. Morale will be very low, mission success probability is down to zero, and safety is highly unknown as the crew and assets are now in the hands of the enemy.

Again, options 1 and 2 are first considered before the last option is taken into consideration since surrender is the last option a fleet commander wants to think of.

### 5.3.2 Simulation Results

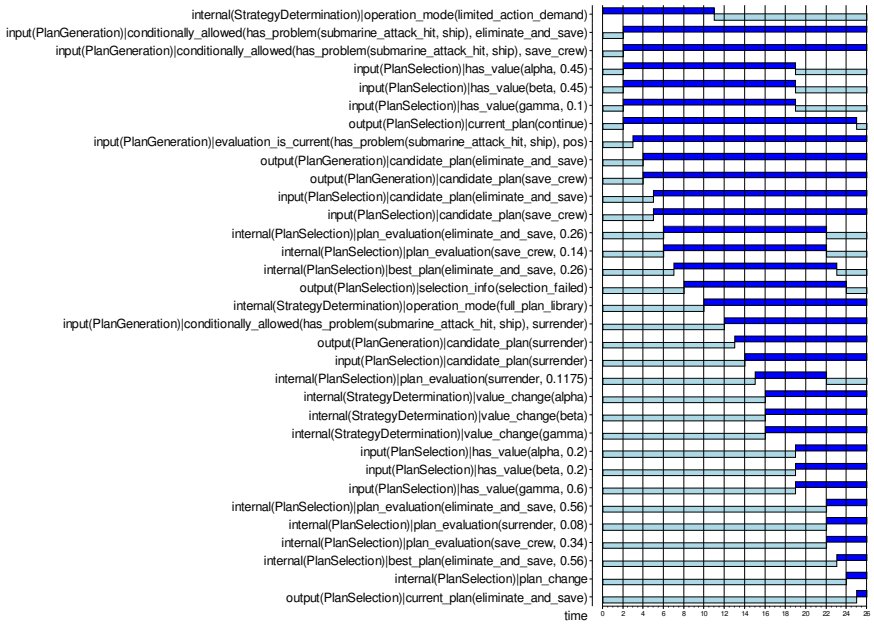


**Fig. 7.** Trace of the frigate loss scenario

Figure 7 shows the simulation results of the Frigate loss scenario. In this particular trace, the α, β, and γ value passed to the PlanSelection component by StrategyDetermination are shown as well. Again, initially the operation mode is set to limited action demand and the accompanying conditional rules for this scenario are passed as well, namely the following:

```
input(PlanGeneration)|conditionally_allowed(has_problem(submarine_attack_hit, ship),
                                                              eleminate_and_save)
input(PlanGeneration)|conditionally_allowed(has_problem(submarine_attack_hit, ship),
                                                                    save_crew)
```

The initial α, β, and γ values passed are respectively 0.45, 0.45, and 0.1:

```
input(PlanSelection)|has_value(alpha, 0.45)
input(PlanSelection)|has_value(beta, 0.45)
input(PlanSelection)|has_value(gamma, 0.1)
```

Denoting that in this case mission success and safety are considered to be more important aspects for plan evaluation than morale. Suddenly the problem of a frigate being hit by an enemy submarine is observed, which is forwarded to the PlanGeneration component:

input(PlanGeneration)|evaluation_is_current(has_problem(submarine_attack_hit), ship), pos)

Based on the detected problem, the two plans that are currently conditionally allowed are generated, and forwarded to PlanSelection:

input(PlanSelection)|candidate_plan(eliminate_and_save)
input(PlanSelection)|candidate_plan(save)

Based on the previously mentioned $\alpha$, $\beta$, and $\gamma$ values, the component evaluates the candidate plans, and concludes that eliminate and save is the best plan, with an evaluation value of 0.26:

internal(PlanSelection)|best_plan(eliminate_and_save, 0.26)

Since the threshold for plan selection is set to a higher value, namely 0.35, the component outputs that selection has failed for this set. As a result the StrategyDetermination component switches to full plan library mode:

internal(StrategyDetermination)|operation_mode(full_plan_library)

The plans that have been added to the library and which are appropriate for the current situation are again forwarded to PlanSelection which evaluates the new additional plan (surrender) to the even lower value of 0.1175:

internal(PlanSelection)|best_plan(eliminate_and_save, 0.26)

Again, selection has failed, however there are no additional plans available in the exceptional action demand mode. Therefore, the StrategyDetermination component decides to adapt the weights of the parameters, and gives more weight to moral ($\gamma$):

input(PlanSelection)|has_value(alpha, 0.2)
input(PlanSelection)|has_value(beta, 0.2)
input(PlanSelection)|has_value(gamma, 0.6)

As a result, the best plan is now eliminate and save which now evaluates above the threshold. Finally, the plan is set to be the current plan.

## 6 Validation by Verification

After the formalized traces have been obtained, (either by formalization of an empirical trace or by means of simulation, such as done in the previous section), it can be validated whether these traces comply to certain desired properties from a more global perspective. Below it is shown which of such properties were identified, in cooperation with domain experts, how they were formally specified. Moreover, verification of these properties against the traces is 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. The first two properties express that the system indeed functions as a meta-level

architecture (as intended), based on upward and downward reflections between the different levels.

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

This property has been automatically checked and shown to be satisfied within the traces. This may sound not too surprising, as the system was designed for this, but this check confirms that what was intended in the design, indeed shows itself in the implementation.

**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($\gamma$, t, output(StrategyDetermination))
|= to_be_assumed(conditionally_allowed(S, P))
$\Rightarrow \exists t2 \geq t$ state($\gamma$, t2, input(PlanGeneration)) |= conditionally_allowed(S, P)]

This property is also satisfied for the given traces.

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

The property is satisfied for the given traces.

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

This property is again satisfied for the given traces.

## 7 Discussion

This paper presents the analysis and simulation of meta-reasoning processes based on an agent-based architecture for strategic planning (cf. [19]) for naval domains. The architecture was designed as a meta-level architecture (cf. [13]) with three levels. The interaction between the levels in this paper is modeled by reflection principles (e.g., [3]). The dynamics of the architecture is based on a multi-level trace approach as an extension of what is described in [9]; see also [5]. The architecture has been instantiated with strategic planning knowledge from the naval domain. As a further contribution, besides mission success, aspects such as safety and fleet morale have been formalized, and incorporated within the strategic knowledge. Without this, it would have been difficult to take such aspects into account in the decision making. Moreover, as discussed in Section 4.3, as plans within the first mode of operation (limited action demand) occur much more frequently than the ones in the second mode (full preferred plan library), a similar relation holds between the second and the third mode of operation (exceptional action demand). As a result of this frequency difference, having such a strategic reasoning level taking this into account, improves the efficiency of the reasoning process

The resulting executable model has been used to perform a number of simulation experiments for different naval scenarios. To evaluate the simulation results and thereby validate the model, in cooperation with domain experts, desired properties for the decision process have been identified, formalized, and verified against the simulation traces.

A meta-level architecture for strategic reasoning in another area, namely that of design processes is described in [7]. This architecture has been used as a source of inspiration for the current architecture for strategic planning. In other architectures, such as in PRS [8], 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 [17] 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 [17]. 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. Furthermore for knowledge-intensive domains, such as the naval domain, there is the issue of scalability. As this heavily depends on the available domain knowledge, only by further exploration for different domains and variants it can be found out how scalable such a system is. It is possible for instance to add or change the described criteria for other domains, but also to apply particular more generic planning algorithms. Finally, this paper addressed resource-bounded situations. In [15] 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 DESIRE [4], future research in simulation and the validation of

relevant properties for the resulting simulation traces is expected to give more insight in the implementation of future complex resource-bounded agent-based planning support systems used by commanders on naval platforms.

**Acknowledgments.** 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. Bosse, T., Jonker, C.M., Meij, L. van der, and Treur, J., LEADSTO: a Language and Environment for Analysis of Dynamics by SimulaTiOn. In: Eymann, T., Kluegl, F., Lamersdorf, W., Klusch, M., and Huhns, M.N. (eds.), *Proceedings of the Third German Conference on Multi-Agent System Technologies, MATES'05*. Lecture Notes in AI, vol. 3550. Springer Verlag, 2005, pp. 165-178.
2. Bosse, T., Jonker, C.M., Meij, L. van der, Sharpanskykh, A., and Treur, J., Specification and Verification of Dynamics in Cognitive Agent Models. In: Nishida, T., Klusch, M., Sycara, K., Yokoo, M., Liu, J., Wah, B., Cheung, W., and Cheung, Y.-M. (eds.), Proceedings of the Sixth International Conference on Intelligent Agent Technology, IAT'06. IEEE Computer Society Press, 2006, pp. 247-254.
3. Bowen, K. and Kowalski, R., Amalgamating language and meta-language in logic programming. In: K. Clark, S. Tarnlund (eds.), *Logic programming*. Academic Press, 1982.
4. Brazier, F.M.T., Jonker, C.M., and Treur, J., Principles of Component-Based Design of Intelligent Agents. *Data and Knowledge Engineering*, vol. 41, 2002, pp. 1-28.
5. Brazier, F.M.T., Jonker, C.M., and Treur, J., Dynamics and Control in Component-Based Agent Models. *International Journal of Intelligent Systems*, vol. 17, 2002, pp. 1007-1048.
6. 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.
7. 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.
8. 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.
9. 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.
10. Hoek, W. van der, Ruan, J., and Wooldridge, M., Strategy Logics and the Game Description Language. In: J. van Benthem, S. Ju and F. Veltman (eds), *A meeting of the Minds*, Texts in Computer Science, College Publications Vol. 8, pp. 259--274, 2007.
11. 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.
12. 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.
13. Maes, P, Nardi, D. (eds), *Meta-level architectures and reflection*, Elsevier Science Publishers, 1988.

14. 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.
15. Pew, R.W. and Mavor, A.S.. *Modeling Human and Organizational Behavior*, National Academy Press, Washington, D.C. 1999.
16. 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.
17. 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.
18. Standaert, D., Tanter, E., Cutsem, T. van, Design of a Multi-Level Reflective Architecture for Ambient Actors. In: *Proceedings of ECOOP Workshop on Object Technology for Ambient Intelligence and Pervasive Computing (OT4AmI 2006),* July 2006, Nantes, France.
19. Treur, J., Formal Semantics of Meta-Level Architectures: Dynamic Control of Reasoning. *International Journal of Intelligent Systems*, vol. 17, 2002, pp. 545-568.
20. Wellman, M. P., Reeves, D. M., Lochner, K. M., Cheng, S.-F., and Suri, R., Approximate strategic reasoning through hierarchical reduction of large symmetric games. In: Twentieth National Conference on Artificial Intelligence, pp. 502–508, 2005.
21. Wilkins, D.E., Domain-independent planning representation and plan generation. *Artificial Intelligence* 22 (1984), pp. 269-301.