Knowledge-Based Systems 22 (2009) 589–599

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

# Agent-based analysis and simulation of meta-reasoning processes in strategic naval planning

Mark Hoogendoorn [a], Catholijn M. Jonker [c], Peter-Paul van Maanen [a,b,*], Jan Treur [a]

[a] Vrije Universiteit Amsterdam, Dept. of Artificial Intelligence, De Boelelaan 1081, 1081 HV Amsterdam, The Netherlands
[b] TNO Human Factors, Dept. of Human Interfaces, P.O. Box 23, 3769 ZG Soesterberg, The Netherlands
[c] Delft University of Technology, Dept. of Man-Machine Interaction, Mekelweg 4, 2628 CD Delft, The Netherlands

ARTICLE INFO

*Article history:*
Received 30 November 2007
Received in revised form 27 October 2008
Accepted 3 May 2009
Available online 10 May 2009

*Keywords:*
Meta-reasoning
Simulation
Planning
Intelligent agent systems

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.



## 1. Introduction

Management of naval organizations aims at maximization of mission success by means of monitoring, planning, and strategic reasoning. The domain of naval operations is characterized by open complex problems, incident prone and resource-boundedness. An incident is an unexpected event, which results in an inappropriate chain of events if left alone. For such domains, plan generation and action selection are important and are supported by strategic reasoning. A typical complex decision is how to deal with an incident; which of the considered plans show most promise, is further investigation necessary, should attention be shifted to different plans. 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 the following important questions. Which plans should be considered, i.e., what criteria are relevant? What criteria should be used for the final plan selection, so that the plan can be executed? Although libraries are in place of possible plans for various situations, every incident is unique. Therefore, the adaptation of plans in the library to the situation at hand takes time. As time is of the essence in incident management, the plan generation process should also be limited. This means that those plans generated first, should have a high probability of mission success, and the criteria for selection should be as sound as possible. Another problem is that not all criteria are easy to measure objectively. For example, mission success, troop morale, and safety of ships and troops are of the utmost importance, but hard to assess and quantify in complex situations. These aspects are introduced and formalized in this paper.

The key selling point of meta-reasoning and meta-level architectures is that they enable systems to reflect on their knowledge state and reasoning process, and therefore can introduce reasonable additional assumptions and make more efficient choices for the reasoning process; see e.g., [3,7,18,19]. This paper goes beyond existing literature in developing a simulation of a real application developed in cooperation with domain experts.

This paper presents a generic agent-based meta-level architecture (cf. [13]) for planning, extended with a strategic reasoning level. Domain experts provided the knowledge and scenarios for the naval domain essential for the development of the system. A conceptual design of the architecture was created using the component-based design method for agent systems DESIRE; cf. [4]. An executable specification was developed in the executable temporal language LEADSTO; see [1] for more details. The knowledge was used to identify and formally specify executable dynamic properties for each of the components within the generic agent architecture. Moreover, 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, see [2]).

* Corresponding author. Address: TNO Human Factors, Dept. of Human Interfaces, P.O. Box 23, 3769 ZG Soesterberg, The Netherlands
*E-mail addresses:* mhoogen@cs.vu.nl (M. Hoogendoorn), C.M.Jonker@tudelft.nl (C.M. Jonker), peter-paul.vanmaanen@tno.nl (P.-P. van Maanen), treur@cs.vu.nl (J. Treur).

0950-7051/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.knosys.2009.05.003

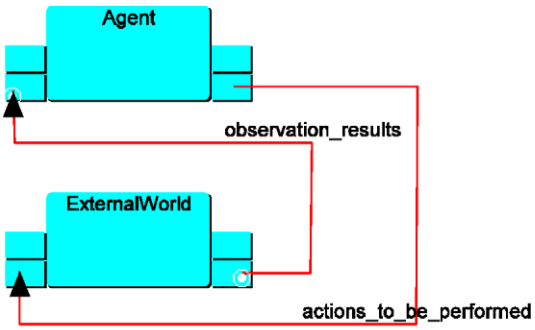


**Fig. 1.** Top-level architecture.

The latter properties have been automatically verified against the simulation traces, 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 TTL Checker tool 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 presented in this section has been specified using the component-based design method for agent systems DESIRE [4]. A comparison of DESIRE with other agent-based modeling techniques, such as GAIA, ADEPT, and MetateM can be found in [14,16]. Note that this architecture lends itself as architecture for a multi-agent system. This paper however only describes the architecture of a single agent. The top-level of the system is shown in Fig. 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 agent model described in [6] of which two components are used: `WorldInteractionManagement` and `OwnProcessControl`, as shown in Fig. 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 lined up for execution by forwarding them to the top-level by means of the information link `actions_from_WIM`. Otherwise, evaluation information is generated and 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` is 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 a figure regarding these components has been omitted.

`OwnProcessControl` is composedof the following components: `StrategyDetermination`, `PlanGeneration`, and `PlanSelection`, see Fig. 3. 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`. Candidate plans are forwarded to `PlanSelection` where the most appropriate plan is selected. In case no plan

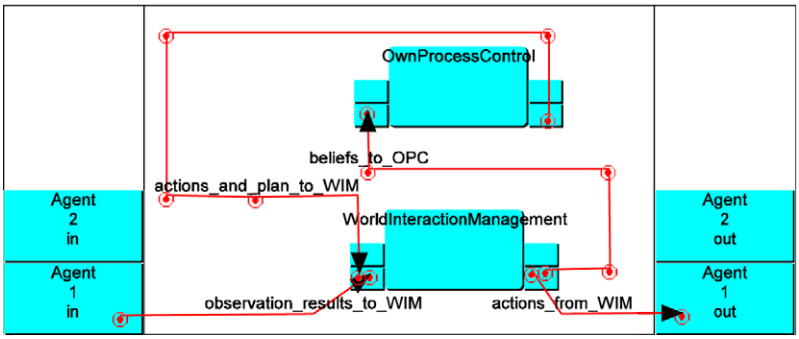


**Fig. 2.** Agent architecture.

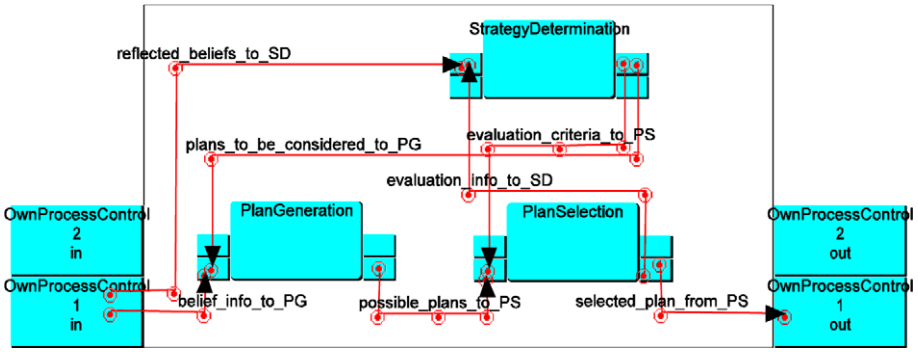


**Fig. 3.** Components within OwnProcessControl.

can be selected in PlanSelection this information is forwarded to the StrategyDetermination component. StrategyDetermination reasons on a meta-level (both the input and output are located on a higher level as shown in Fig. 3). Its input consists of reflected beliefs (originating in WorldInteractionManagement) and of the status of the plan selection process from PlanSelection. The component has the possibility to generate extra 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 is on average more efficient, as the initial number of options is limited. Note, that efficiency thus depends on having the most straightforward options available at the object level.

## 3. Formalization method

This section presents the method used to develop the formalizations presented in Sections 4 and 6. The conceptual model presented in Section 2 is formalized in a detailed model in Section 4. The dynamics properties used to validate the system are presented in Section 6. The dynamic properties essential in naval strategic planning are such that an expressive language is required. The Temporal Trace Language (TTL) is used as a tool for specifying properties that need validation; cf. [12]. LEADSTO is chosen for simulation. The following provides a brief overview of TTL, and LEADSTO. These two languages have been aligned to enable easy checking of TTL properties against simulations in LEADSTO.

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* $DYNPROP(\sum)$ 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 $\gamma$ 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($\gamma$, t, input(c)).

Analogously state($\gamma$, t, output(c)) and state($\gamma$, 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($\gamma$, t, output(c)) |= p denotes that state property p holds in trace $\gamma$ 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($\gamma$, t, output(c))|= p are shortened to output(c)|p.

To model direct temporal dependencies between two state properties, the simpler LEADSTO format is used. This is an executable format defined as follows. Let $\alpha$ and $\beta$ 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 LEADSTO language $\alpha \rightarrow\!\!\!\!\rightarrow_{e,f,g,h} \beta$, means:

If state property $\alpha$ holds for a certain time interval with duration g, then after some delay (between e and f) state property $\beta$ 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 LEADSTO format has as advantage that it is executable and that it can easily be depicted graphically.

## 4. Component specification for naval planning

This section introduces those components within the strategic planning process in more detail that are most relevant for the planning process, i.e., the components of OwnProcessControl. A partial specification of executable properties in formal format is 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 generally as the knowledge about plans. Conditions for those plans are stored in the StrategyDetermination component, described in Section 4.3. 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)
```

This rule expresses that in case component Monitoring evaluated the current situation as being situation S and PlanGeneration has received an input that situation S allows for plan P then plan P 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. 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. For the sake of brevity details about the bridge between changes of the criteria after an observation and the overall value of the criteria are not formalized.

#### 4.2.1. Mission success

This obviously important criterion is related to the objective of the mission. In case a decision needs to be made, its influence on

mission success needs to be determined. The criterion refers to several factors regarding deadlines, and fleet configurations. The first considers the probability that the deadline is reachable. The second considers the probability that the mission succeeds with a specific fleet configuration. The combined probability value of mission success is a real number between 0 and 1. The impact of possible events on mission success was identified by a naval domain expert. The impact 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. Changes in the situation lead to changes 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)
```

#### 4.2.2. Safety

Safety is a complex criterion as well. When a ship loses propulsion the probability of survival of the ship decreases dramatically if it is left alone by the others. Not leaving it behind might endanger the safety of the others. Basically, the probability of survival depends on three factors: (1) the speed with which the task group is sailing; (2) the configuration of the fleet, 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(S:REAL * 0.5)
```

#### 4.2.3. Fleet morale

The morale of the men on board of the ships is also important as troops with good morale are much more likely to win compared to those with poor 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 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 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: to determine the conditional plans that are to be used given the current state, and to provide a strategy for selecting one of these plans.

In general, naval plans are generated according to a preferred plan library. Only in exceptional cases are plans generated that fall 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 which of three modes of operation is to be used in the current situation, and therefore 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 and 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. Section 4.2 states that plan selection depends on mission success, safety, and fleet morale. All three factors have an impact on the overall evaluation of a plan. 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):

$$\texttt{evaluation\_value(P)} = \alpha * \texttt{mission\_success\_value(P)} + \beta * \texttt{safety\_value(P)} + \gamma * \texttt{fleet\_morale\_value(P)}$$

where all values and degrees are in the interval [0,1], $\alpha + \beta + \gamma = 1$, and P is a plan.

In a rule format, this is reflected in the following general rule:

```
if   mission_succes_weight(Alpha:REAL)
  and   safety_weight(Beta:REAL)
  and   fleet_moral_weight(Gamma:REAL)
  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(P:PLAN, Alpha:REAL * R1:REAL + Beta:REAL * R2:REAL + Gamma:REAL * R3:REAL)
```

The weights depend on the type of mission and the current state of the process. 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)
then  mission_succes_weight(0.34)
  and  safety_weight(0.33)
  and  fleet_moral_weight(0.33);
```

However, if a mission is supposed to be executed safely at all cost or the situation shows that already many ships have been lost, the weight for safety should be relatively high.

```
if  problem_type(mission_success_important)
  and  problem_type(safety_important)
  and  not problem_type(fleet_morale_important)
then  mission_succes_weight(0.45)
  and  safety_weight(0.45)
  and  fleet_moral_weight(0.1);
```

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)
then  mission_succes_weight(0.6)
  and  safety_weight(0.2)
  and  fleet_moral_weight(0.2);
```

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 Sections 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 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 Fig. 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:

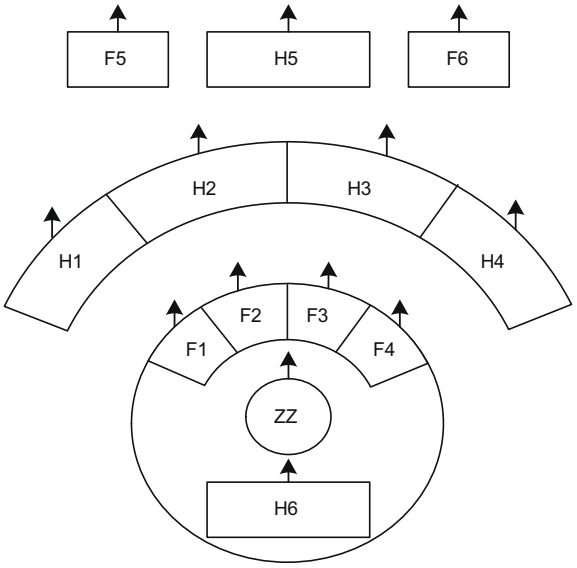


**Fig. 4.** Scenario for meta-reasoning

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 Sections 2 and 4, and instantiated with the case-study specific knowledge from Section 5.1 are shown in Fig. 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 readable 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 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

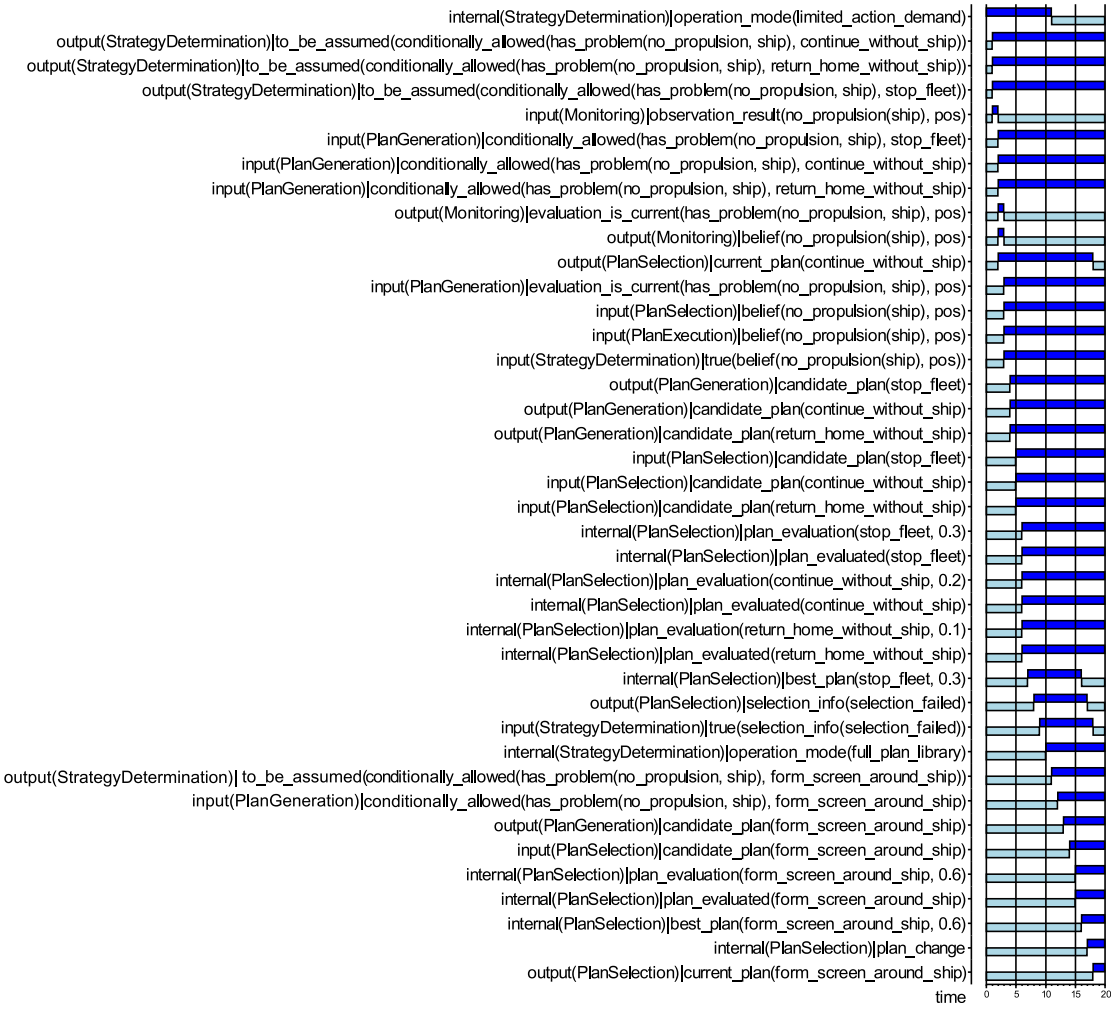


**Fig. 5.** Trace of the total steam failure simulation.

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_propul-
  sion(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_pro-
  pulsion, 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 (ZZ) 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 ZZ. 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 ZZ is no longer protected, and another enemy ship could possibly attack ZZ. The risk for mission success is therefore high, and morale of the men on board of the ships part of ZZ 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 ZZ. 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 ZZ. 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 plan to eliminate and turn.

#### 5.2.2. Simulation results

Fig. 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, on average this saves a lot of precious evaluation 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


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

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 ZZ. 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 shows the simulation results of the Frigate loss scenario. In this particular trace, the criteria weights 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 criteria weights passed are respectively 0.45, 0.45, and 0.1:

```
input(PlanSelection)|has_value(mission_succes_
   weight, 0.45)
input(PlanSelection)|has_value(safety_weight, 0.45)
input(PlanSelection)|has_value(fleet_moral_weight,
   0.1)
```

This denotes 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 criteria weights, 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)
```

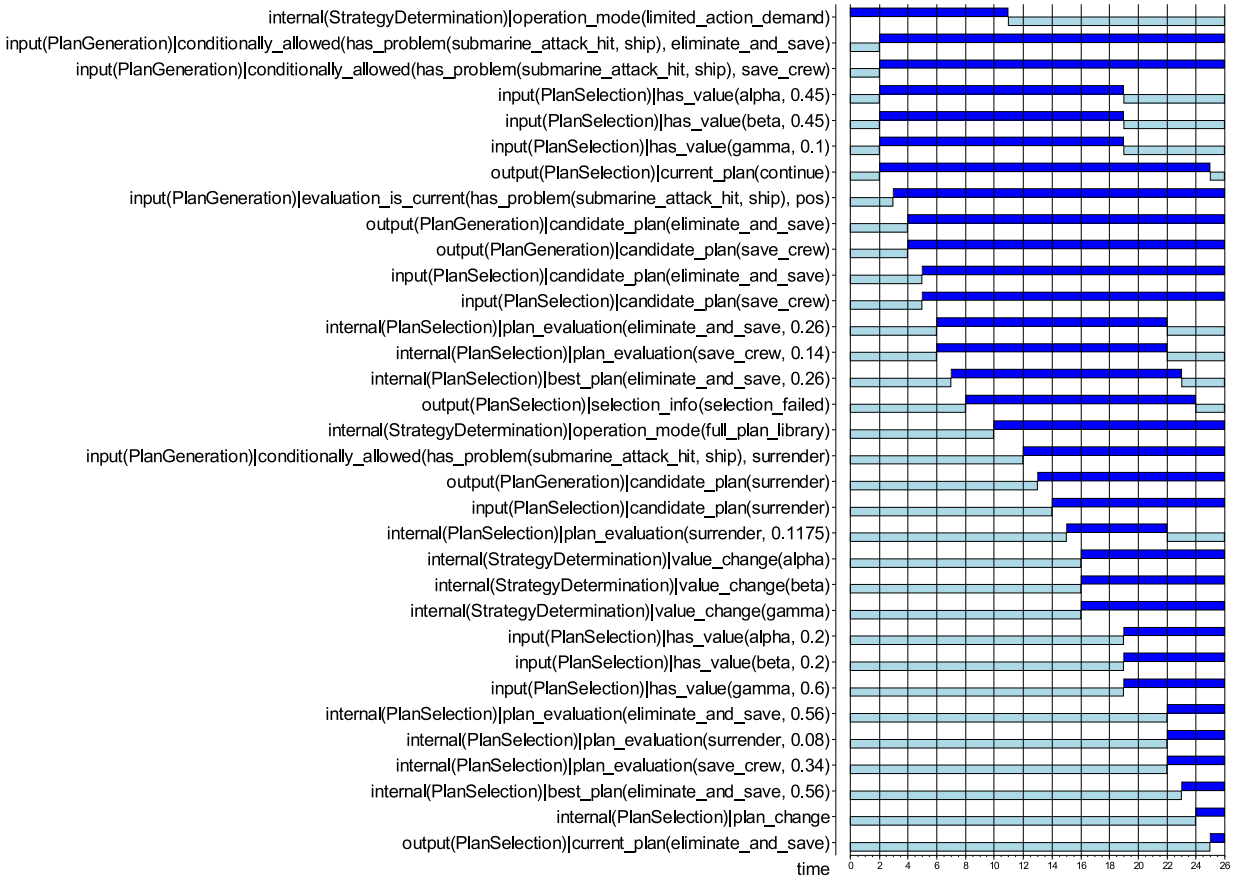


**Fig. 7.** Trace of the frigate loss scenario.

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(elimi-
  nate_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:

```
input(PlanSelection)|has_value(mission_succes_
  weight, 0.2)
input(PlanSelection)|has_value(safety_weight, 0.2)
input(PlanSelection)|has_value(fleet_moral_weight,
  0.6)
```

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

## 6. Validation by verification

Formalized traces can be obtained by formalization of an empirical trace or by means of simulation, such as done in the previous section. The next step is to validate whether the traces comply with the desired properties from a more global perspective. This section presents the properties identified for this purpose and their formal specification. The properties were identified in cooperation with domain experts (naval officers). 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 Section 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 $\geqslant$ 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 $\geqslant$ 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(Strategy Determination)) |= to_be_assumed(conditionally_allowed(S, P)) $\Rightarrow\exists$t2 $\geqslant$ 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′ $\geqslant$ t & t′ $\leqslant$ 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 describes that a plan is only changed in case there has been an exception that triggered this change. Formally:

$\forall$t, t2 $\geqslant$ 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′ $\geqslant$ t & t′ $\leqslant$ 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 for strategic planning in naval domains. The simulation has been developed on the basis of an agent-based architecture where the internal agent 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 are 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, the impact on planning of such complex criteria as mission success, safety and fleet morale have been formalized, and incorporated within the strategic knowledge. The three leveled approach corresponds with three levels of operation: the first mode of operation (limited action demand) occurs much more frequently than the second mode (full preferred plan library), and a similar relation holds between the second and the third mode of operation (exceptional action demand). As a result of this frequency difference, the strategic reasoning approach presented in this paper improves the efficiency of the reasoning process.

Although the generation of plan libraries is done offline, in open and complex environments, such as the naval domain, precompiled preferred plan libraries are not always sufficient. The meta-reasoning architecture presented in this paper still applies in such situations, and can be combined with approaches to dynamically form plan libraries. For example, one can initiate strategic planning based on precompiled plan libraries, and when all criteria are not satisfied, one can fall back on (less efficient, but possibly more complete) conventional planning, taking the previous failures into account.

The executable model described in this paper has been used to perform a number of simulation experiments for different naval scenarios. To evaluate the simulation results and thereby validate the model, properties for the decision process have been identified, formalized, and verified against the simulation traces. These properties have been identified in cooperation with domain experts (naval officers). Before it can be used in practice, future efforts should also focus on validation, verification, and acceptance of the architecture by a broader group of naval officers and officials.

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 outcomes 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.

For knowledge-intensive domains, such as the naval domain, there is the issue of scalability. Testing for scalability requires domain knowledge. Only by further exploration for different domains and variants can such knowledge be created. Such exploration would also allow for testing the system with other criteria for plan generation and selection and with more generic planning algorithms.

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.

## Acknowledgement

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] T. Bosse, C.M. Jonker, L. van der Meij, J. Treur, LEADSTO: a language and environment for analysis of dynamics by SimulaTiOn, in: T. Eymann, F. Kluegl, W. Lamersdorf, M. Klusch, M.N. Huhns (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] T. Bosse, C.M. Jonker, L. van der Meij, A. Sharpanskykh, J. Treur, Specification and verification of dynamics in agent models, International Journal of Cooperative Information Systems 18 (2009) 167–193.
[3] K. Bowen, R. Kowalski, Amalgamating language and meta-language in logic programming, in: K. Clark, S. Tarnlund (Eds.), Logic Programming, Academic Press, 1982.
[4] F.M.T. Brazier, C.M. Jonker, J. Treur, Principles of component-based design of intelligent agents, Data and Knowledge Engineering 41 (2002) 1–28.
[5] F.M.T. Brazier, C.M. Jonker, J. Treur, Dynamics and control in component-based agent models, International Journal of Intelligent Systems 17 (2002) 1007–1048.
[6] F.M.T. Brazier, C.M. Jonker, J. Treur, Compositional design and reuse of a generic agent model, Applied Artificial Intelligence Journal 14 (2000) 491–538.
[7] F.M.T. Brazier, P.H.G. van Langen, J. Treur, Strategic knowledge in design: a compositional approach. knowledge-based systems, in: K. Hori (Ed.), Special Issue on Strategic Knowledge and Concept Formation, vol. 11, 1998, pp. 405–416.
[8] M.P. Georgeff, F.F. Ingrand, Decision-making in an embedded reasoning system, in: Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89), Detroit, MI, 1989, pp. 972–978.
[9] W. van der Hoek, J.-J.Ch. Meyer, J. Treur, Formal semantics of meta-level architectures: temporal epistemic reflection, International Journal of Intelligent Systems 18 (2003) 1293–1318.
[10] W. van der Hoek, J. Ruan, M. Wooldridge, Strategy logics and the game description language, in: J. van Benthem, S. Ju, F. Veltman (Eds.), A Meeting of the Minds, Texts in Computer Science, vol. 8, College Publications, 2007, pp. 259–274.
[11] C.M. Jonker, J. Treur, A compositional process control model and its application to biochemical processes, Applied Artificial Intelligence Journal 16 (2002) 51–71.
[12] C.M. Jonker, J. Treur, Compositional verification of multi-agent systems: a formal analysis of pro-activeness and reactiveness, International. Journal of Cooperative Information Systems 11 (2002) 51–92.
[13] P. Maes, D. Nardi (Eds.), Meta-Level Architectures and Reflection, Elsevier Science Publishers, 1988.
[14] M. Mulder, J. Treur, M. Fisher, Agent modelling in metateM and DESIRE, in: M.P. Singh, A.S. Rao, M.J. Wooldridge (Eds.), Intelligent Agents IV, Proceedings Fourth International Workshop on Agent Theories, Architectures and Languages, ATAL'97, Lecture Notes in AI, vol. 1365, Springer Verlag, 1998, pp. 193–207.
[15] R.W. Pew, A.S. Mavor, Modeling human and organizational behavior, National Academy Press, Washington, DC, 1999.
[16] O. Shehory, A. Sturm, 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] J. Sokolowski, 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] D. Standaert, E. Tanter, T. van Cutsem, 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] J. Treur, Formal semantics of meta-level architectures: dynamic control of reasoning, International Journal of Intelligent Systems 17 (2002) 545–568.
[20] M.P. Wellman, D.M. Reeves, K.M. Lochner, S.-F. Cheng, R. Suri, Approximate strategic reasoning through hierarchical reduction of large symmetric games, in: Twentieth National Conference on Artificial Intelligence, 2005, pp. 502–508.
[21] D.E. Wilkins, Domain-independent planning representation and plan generation, Artificial Intelligence 22 (1984) 269–301.