Proceedings of the 2013 Winter Simulation Conference
R. Pasupathy, S.-H. Kim, A. Tolk, R. Hill, and M. E. Kuhl, eds.

A STOCHASTIC DISCRETE EVENT SIMULATOR FOR EFFECTS-BASED PLANNING

Hirad Asadi
Johan Schubert

Department of Decision Support Systems
Swedish Defence Research Agency
SE-16490 Stockholm, SWEDEN

ABSTRACT

In this system oriented paper we describe the architectural framework and information flow model of a stochastic discrete event simulator for evaluating military operational plans. The simulator is tailored for Effect-based Planning where the outcome of a plan is compared with a desired end state. The simulator evaluates several alternative plans and identifies those that are closest to the desired end state. As a test case we use a scenario which has been developed by the Swedish Armed Forces in their Combined Joint Staff Exercises. The scenario is carried out in a fictitious country called Bogaland. The simulator focuses on separation of military scenario data (implemented as an XML-model) and military action logic (implemented as a rule based engine). By separating scenario data from actors' behavior rules the modeling task becomes easier for subject matter experts. The results show that alternative plans can be identified based on efficiency and effectiveness.

1 INTRODUCTION

Effect-based military operational planning is a well-known concept explained among others in (Hunerwadel 2006). In Schubert et al. (2010) we examine how high-level military plans can be modeled in order to identify the best plan alternatives. However, in this paper we focus mainly on the architectural features and information flow model of the stochastic discrete event simulator developed at the Swedish Defence Research Agency, FOI. The simulator enables a military analyst to identify the best military plans (chain of actions) among a large amount of possible action alternatives. The output data that are produced by the simulator indicates the most efficient and effective plans, and can also be used for deeper analysis tasks within data farming (Horne and Meyer 2010).

Effects-based Planning (EBP) is a part of a thought process called an Effects-based Approach to Operations (EBAO). The idea is to have an "effects-based thinking" when designing military operations in order to reach certain effects or desired end states (Hunerwadel 2006). EBAO in general is composed by four parts: EBP for creating military plans, Effects-based Execution (EBE) which focuses on executing those plans, Effects-based Assessment (EBA) which deals with following up on the executed plans and knowledge support which provides background knowledge to the previous parts (Schubert et al. 2010).

The simulator focuses on analyzing the EBP process. The general idea behind this research is that if there are many possible combinations of actions for a military plan (alternatives consisting of actions), then a simulator can evaluate many of these combinations in order to show the decision maker which possible combinations are most successful.

The general outline of the system architecture developed in this research is shown in Figure 1. The main system components are:

- Plan XML-Model,
- Plan Instantiator,

- Simulation Engine,
- Behavior Engine,
- Military Domain Rules,
- Analysis Classes.

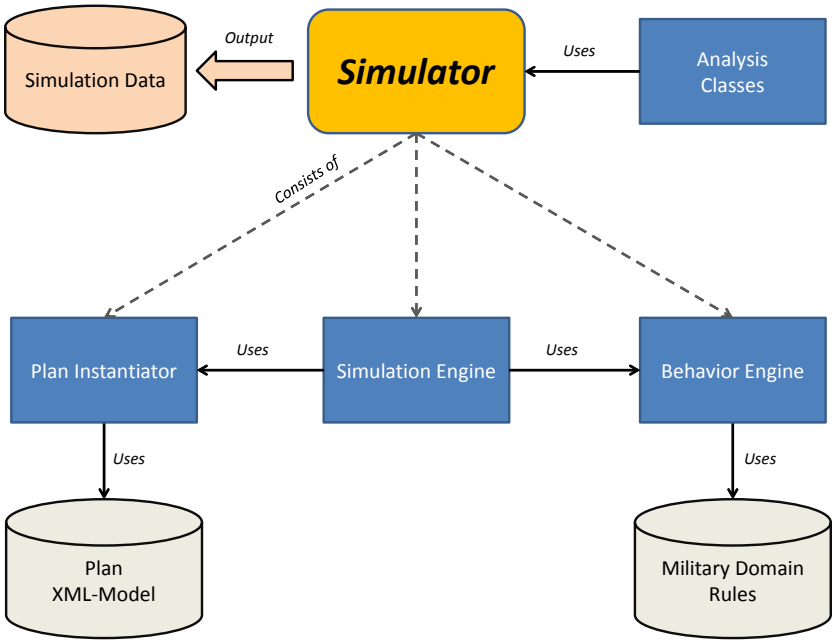


Figure 1: System architecture.

Each component has a specific role which is explained in the upcoming sections. The system utilizes scenario specific military data which are stored as an XML model (Ligeza 2001), called the Plan XML-Model. The data are initialized as runtime objects by the Plan Instantiator. What this means is that each action along with its involved actors reside in memory. The data structure which holds the actions is a tree structure, where all alternatives to an action exist on the same level in the tree. When the data have been instantiated the model can be accessed by the Simulation Engine. The problem of achieving the end state of EBP corresponds to finding the best sequence of action alternatives. This can be represented as a search tree with actions as nodes and alternatives as branches. As this representation corresponds to the representation used in the A* algorithm we will use A* to solve the problem. Thus, the Simulation Engine uses the A* algorithm (Hart, Nilsson, and Raphael 1968) to find the best action combinations by executing actions and comparing the distance costs calculated using a distance function as shown in Equation 1. We use the A* algorithm because it achieves better time performance compared to normal Dijkstra's shortest path algorithm (Dijkstra 1959). In general, the purpose of the A* algorithm is to find the path with the shortest distance from a starting point to a specific goal. This can be achieved by having knowledge about the distance traveled and an estimate about the remaining distance that lies ahead. When an action has been executed, actor responses are carried out by the Behavior Engine, a rule-based component which utilizes specific military scenario logic (Military Domain Rules) that are separately stored in text files.

As a test case we use a scenario which has been developed by the Swedish Armed Forces in their Combined Joint Staff Exercises. The scenario is carried out in a fictitious country called Bogaland. This country consists of different territories which are inhabited by a variety of actors (40 actors in total). The actors are classified as hostile, friendly, neutral or uncertain towards the "blue forces", which represent the military planner. These classifications are predefined by the subject matter expert but can be changed

dynamically during simulation execution. The classifications are calculated using actor parameters and actor relations towards other actors. When the parameters and relations change, the classification of an actor is updated based on subject matter expert rules (Military Domain Rules).

Bogaland holds many conflicts that have their roots in old wars and historical clashes between different groups in different regions. These conflicts involve the control and exploitation of natural resources, religious divergences and many other issues. The idea of this scenario is to create a place which includes many real world conflicts. In this paper we use this scenario and create a data model that aims to represent a military plan for minimizing the impact of hostile actors in specific regions in Bogaland. What this means is that certain hostile actors are in focus and for those actors it is necessary that different parameters must be changed according to a goal. For example, it is important to minimize "weapon power" for all hostile actors.

Simulating alternative plans can be done in different ways depending on which level of abstraction (operational, tactical, etc.) you are interested in. Different government agencies have chosen different strategies when creating decision support tools for planning of military operations. For example, one approach to simulate alternative plans is to use qualitative reasoning which is done in SimPath (Hinrichs et al. 2011). SimPath is a hybrid simulator under development by DARPA's Deep Green program (Surdu and Kittka 2008). The idea behind SimPath is to combine qualitative reasoning, a geographic information system and targeted probabilistic calculations in order to simulate alternative plans. Among other things, SimPath shows the plans that are acceptable, partially acceptable and unacceptable.

A third approach to simulation-based decision support is demonstrated by the Peace Support Operations Model (PSOM). This is a framework developed by the UK Ministry of Defence for peace supporting operations (Body and Marston 2010). The model uses a human-in-the-loop approach to deal with situation stabilization during irregular warfare. PSOM has a three-stage review process where the first stage deals with subject analysis (e.g., data search and evaluation). The second stage deals with specification (e.g., creation of conceptual models) and the third stage is application (code development, verification, testing and review); "The PSOM is an attempt to examine the conduct of a campaign by placing each component of the operation in context, describing the constructive or destructive relationships between them" (Body and Marston 2010).

The outline of this paper is as follows. In Section 2 we explain the data model (military plan), which is fed into the simulator. After this, more detailed information about the simulator components are explained in Sections 3-4. In Section 5 we describe how the simulator can be used by analysis classes. Finally, we show the results in Section 6 and end with conclusions in Section 7.

2 PLAN XML-MODEL

In order to store scenario data we use the XML storage format since it is flexible, simple and widely used. The goal is to capture all scenario concepts that might be of interest without introducing any extra complexity in the model. Since EBAO is a high-level thought-process we only model concepts that are absolutely necessary for understanding the scenario. The main concepts that we model using XML-elements are:

- actor,
- action,
- relation.

An actor can represent a physical individual or an aggregated unit. For example, it can be a single tank or a whole battalion. The *actor* element contains information about the actor, such as *name*, *description* and actor *parameters* which are different abilities rated on a scale from 0 to 3 (Schubert et al. 2010) as shown in Figure 2. These 15 parameters are associated with a target value (*goal* attribute) which indicates what the desired end state should be for each *parameter* for every *actor*. These *goal* values are used by

the Simulation Engine to calculate how far we are from the sought after end state using a specific distance function (Schubert et al. 2010). In order to acquire the distance between two subsequent actions a_x and a_y we calculate the sum of the difference between each actor parameter p_{ij}^x after action a_x and the same (updated) actor parameter p_{ij}^y after action a_y as shown in Equation 1. To find the distance from a_x to the end state we replace a_y with the sought after end state.

```
- <actor id="3" name="EK reg 3" x="6696035" y="1582325" areax="6696035" areay="1582325"
    areaw="50000" areah="190000">
  - <parameters>
      <parameter name="weaponpower" goal="1">2</parameter>
      <parameter name="livingconditions" goal="2">2</parameter>
      <parameter name="stance" goal="1">2</parameter>
      <parameter name="sympathizers" goal="1">3</parameter>
      <parameter name="economy" goal="1">1</parameter>
      <parameter name="stability" goal="2">2</parameter>
      <parameter name="geographicaldominance" goal="1">3</parameter>
      <parameter name="infrastructure" goal="1">2</parameter>
      <parameter name="propagandachannels" goal="1">2</parameter>
      <parameter name="socialnet" goal="2">3</parameter>
      <parameter name="reputation" goal="1">3</parameter>
      <parameter name="dissatisfaction" goal="0">0</parameter>
      <parameter name="groupfeeling" goal="2">2</parameter>
      <parameter name="idelogicalconviction" goal="1">3</parameter>
      <parameter name="goalorientation" goal="1">2</parameter>
    </parameters>
  </actor>
```

Figure 2: Example of an actor XML-element.

$$d(a_x, a_y) = \sum_{i=1}^{40} \sum_{j=1}^{15} |p_{ij}^x - p_{ij}^y|, \tag{1}$$

where i is an index over all 40 actors and j is an index over all 15 parameters for each actor.

The *action* element contains *name*, *description* and a *group* attribute that describes which level in the action tree the action occurs at as shown in Figure 3. In addition, the action element contains data about which actors are involved in the action. In Figure 3 we can see that this particular action has a description which says "neutralize". This means that specific targets must be neutralized, in this case an actor called "DSD", "KSP" and other irregular units. Both the *actor* and *action* elements have a special attribute called *class*, which indicates what implementation class needs to be used for that particular element during instantiation. There are other attributes associated with the elements as well, such as geographical coordinates. These elements are not used by the simulator, but may be of interest for visualization purposes.

```
- <action id="71" group="15" name="Neutralise DSD, KSP and Other IRR units" class="NeutralizeActivity"
    description="Neutralise DSD, KSP and Other IRR units By Negotiating And buy out compliance"
    time_start="85" time_stop="160" x="6696035" y="1582325" areax="6696035" areay="1582325"
    areaw="50000" areah="190000">
  + <actorSet>
  </activity>
```

Figure 3: Example of an action XML-element.

The *relation* element is used to initiate actors with relational data about other actors, e.g., if actor A is friendly towards actor B, then this is indicated in the *relation* element. Note that the data change dynamically during runtime. Figure 4 shows each relation that actor with id "39" has.

Each color represents a type of relation in the XML model, red means hostile, yellow stands for uncertain, blue is friendly and green is neutral. It is the Behavior Engine's responsibility to update the relation data. This is achieved by taking into account the current executing actor responses and the current actor states along with different sets of rules defined by a subject matter expert.

```
- <relation actorid="39">
    <rel id="40" color="red" />
    <rel id="13" color="yellow" />
    <rel id="14" color="yellow" />
    <rel id="15" color="green" />
    <rel id="16" color="yellow" />
    <rel id="17" color="yellow" />
    ...
```

Figure 4: Example of a relation XML-element.

3 SIMULATOR

The simulator uses discrete events (Allen 2011) for simulating military operative plans. The discrete events, namely the actions and actor responses, occur as a sequence in time. Each simulation equals one completed action (i.e., one node) in the action tree. Since an action is complex, involving many actors with parameters which are updated during execution, we need to store all this information if we want to restart from the simulation end-point in the future. This storage of all parameters, or snapshot of the "system state", is important because the A* algorithm can come to a certain node when it is no longer feasible to continue execution on that same branch. Hence, it must backtrack to a previously discovered action and choose new alternatives from there. By storing system states we can directly jump from one sequence of action alternatives to another using the exact same parameter settings which were valid for a previously evaluated sequence of action alternatives. If we do not store the system state for each action we can not backtrack because the parameter data will be overwritten for each new action execution.

Before explaining the components involved we need to describe the data flow of the simulator. Figure 5 shows how data are loaded into the simulator which then executes and outputs processed data.

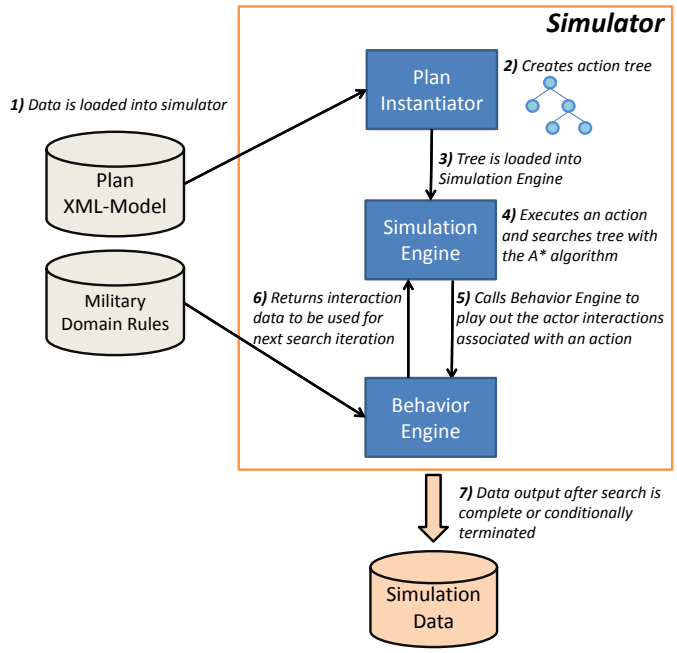


Figure 5: Simulator flow chart.

The XML model is loaded into the Plan Instantiator (Step 1) which creates an action tree (Step 2). The action tree is loaded into the Simulation Engine (Step 3) which uses this tree to perform searches (Step 4). When an action is executed the main execution task falls upon the Behavior Engine to perform scenario interactions (Step 5) which utilizes rules from the military domain. An alternative sequence of activities

is created since when a tree level consists of n activities and the next level consists of m activities, there are $n \times m$ combinations just for those two levels.

When the Behavior Engine has finished performing the interactions the data are sent back to the Simulation Engine (Step 6) which uses it in order to calculate where to continue its search along the tree. Finally, when the search is complete or a certain condition is met, the simulator terminates and outputs the results (Step 7).

3.1 Plan Instantiator

The role of the Plan Instantiator is simply to parse the XML model and create object instances. It achieves this by reading the XML elements and instantiating objects based on the *class* attribute. When reading the action elements the Plan Instantiator takes into account the *group* attribute of an action and creates a tree based on it. The tree data along with actor information and relations are stored for further usage by the Simulation Engine.

3.2 Simulation Engine

The Simulation Engine is the core component of the simulator and is focused on the actions of the plan. It uses the tree data from the Plan Instantiator to perform A* search as shown in Figure 6. Normally the A* algorithm searches and stores single nodes in a queue, but since we are interested in storing paths, for traceability, we have to extend the algorithm to support this feature. Each path has its own calculated distance costs which are used for comparison between paths. The A* algorithm searches those paths that have the best (least distance) costs from a queue.

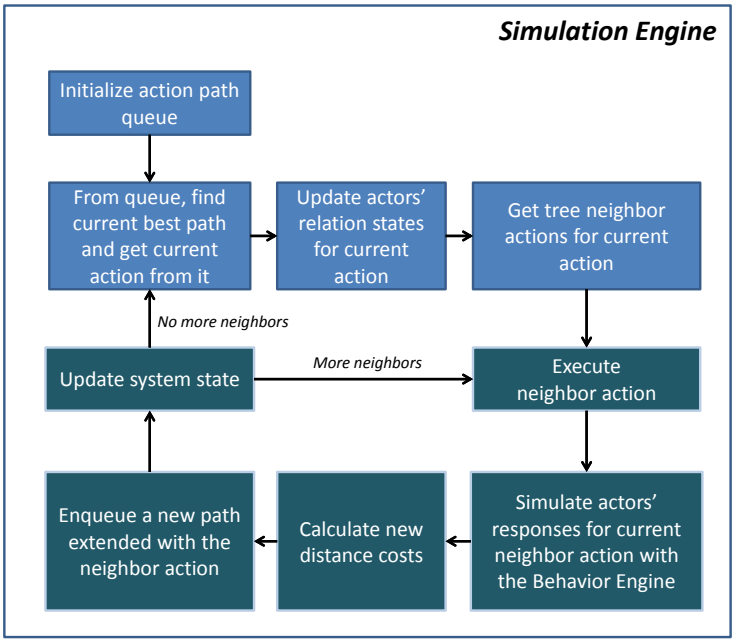


Figure 6: Simulation Engine flow chart.

Because our scenario contains 2.1×10^{23} possible combinations we also need to tell the A* algorithm to terminate based on a condition in order to avoid memory overflow. The number of alternative plans is calculated as the product of alternatives for each action. We do this by limiting the size of the sorted queue which is used by the A* algorithm to 10 000 elements. What this does is it allows the queue to only be expanded by paths that are better than preexisting ones. The problem is that we might lose special cases

where the initial cost of selecting an action is high, but the cost after that action has been selected is less compared to the best path found so far.

The first thing we do is to initialize the queue with a new path consisting of the root action and calculate distance costs. We then enter the main A* algorithm loop and start dequeuing the best path, which has the least distance cost.

When finding the current best path from the queue we look at the latest action in it and start updating all actor relations associated with that particular action. When this is done we get all neighbors for that current action (according to the A* algorithm). For each neighbor action we execute it and call the Behavior Engine. The Behavior Engine returns the actor responses based on the current executed neighbor action and associated actors. Based on the received response we calculate new distance costs and enqueue a new path which has been extended with the current neighbor. Finally, the system state is updated and if there are any more neighbors we continue executing them according to Figure 6, otherwise we continue dequeuing the next best path until the queue is empty or a maximum number of complete plans have been evaluated.

3.2.1 System State Renewal

The system state indicates the state of all actors, namely, their quantitative parameters and relations. When a state renewal takes place all 600 state variables (40 actors with 15 parameters each) are updated based on distribution data that have been calculated by the Behavior Engine (using the Monte Carlo method) as shown in Section 3.3. The distribution is over actor parameter values. For each actor and all actor parameters a frequency table is created of how many times a certain value occurred. The distribution data are used by the update routine to calculate and set new values for the actor parameters and relations. When the system state has been updated the new actor values are used by the Simulation Engine for the next iteration (Schubert et al. 2010).

3.2.2 Simulation Output Data

During the simulation different data are gathered. The most important data are the sequences of actions that have been discovered by the simulator. These sequences indicate which combinations of actions that are most efficient and effective among the 2.1×10^{23} alternatives. Each combination contains a sequence of actions together with the cost for selecting that particular sequence. The costs consist of three different values (distances) according to the A* algorithm: the f-value (total distance), the g-value (distance traversed) and the h-value (estimated distance to goal). The f value is simply $g+h$ and all distances are calculated from a specific action's perspective. This means that each action has its own set of distance values. Note that even though the traversed or total distance increases, this does not mean we are closer to the goal since the h-value solely is an indication of how close we are to the goal. Minimizing h corresponds to approaching the end state of EBP.

The minimum h-value is 0, but in a real world scenario you can never achieve this simply because we cannot satisfy every requirement of the end state. The maximum distance to goal is when all actor parameters have values which are as far away as possible from their goal values. For example, with a total of 600 parameters and 3 as the most divergent value for each parameter, the maximum distance is 1800 $(600 * 3)$ according to the distance function in Equation 1. The smaller the h-value is, the closer we are to the goal. From an EBP perspective we can use the h-value for defining effects, e.g., since we know the maximum h-value possible we can calculate to what percentage the goal is achieved for each actor. We also want to achieve traceability, therefore all parameter values are sampled every time we execute a new action.

3.3 Behavior Engine

The purpose of the Behavior Engine is to generate actor responses based on the current executed action. The responses are gathered in the form of a parameter distribution which is sent back to the Simulation Engine for further processing as shown in Figure 7.

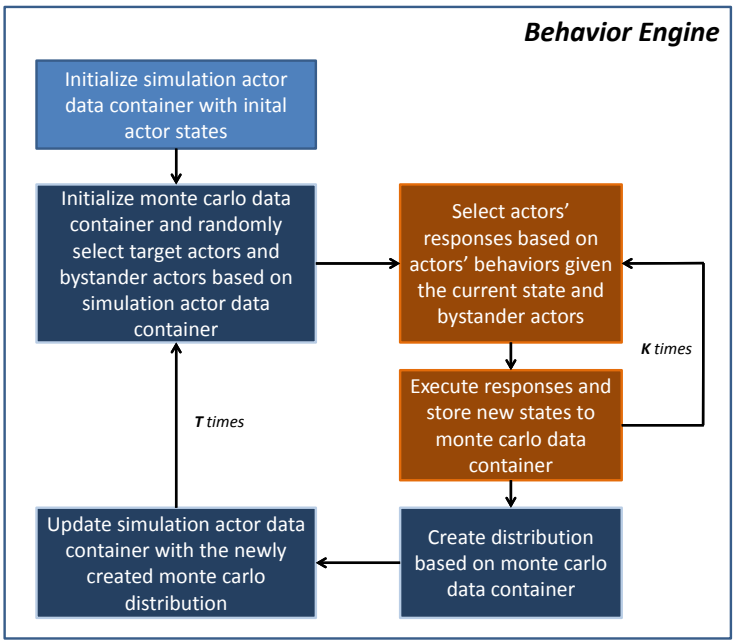


Figure 7: Behavior Engine flow chart.

The Behavior Engine uses a rule-based system inside the Monte Carlo method for calculating actor responses. At first, the actor states are initialized into a global actor data container which will be used by the Behavior Engine. In the Monte Carlo method, a new Monte Carlo data container is created for each Monte Carlo iteration. First, the algorithm selects target actors and bystanders randomly. After this the algorithm selects the actors' responses based on their behavior given the circumstances, such as current executed action and relations. Secondly, the actor responses are executed and the resulting data are stored in the Monte Carlo container, previously created.

This process of executing actor responses is repeated for a specific maximum number of times (K times). The higher the value, the more times an actor is allowed to react before the initiation of the next action of the plan. Since this is a trade-off criteria between computation time and model accuracy (and error propagation), an analyst has to decide what values should be used. When the response execution has been finished a distribution is created based on the Monte Carlo data container. The Monte Carlo method does T replications before terminating. In our experiment we use $T = 20$ and $K = 2$. When all is finished the actor data container is returned to the Simulation Engine which uses it in order to calculate new distance costs and finally update the system state.

The actor model is based on the actor state variables which have integer values from 0 to 3. This information is used to create the behavior model for each actor, which in turn controls how an actor reacts. The behavior model is based on a Bayesian network (Pourret, Naim, and Marcot 2008) which uses subject matter expert knowledge and machine learning algorithms (Luotsinen and Sjöberg 2012).

With this type of model an actor's response affects its state variables and relations. The state variables may also be dependent on each other or on the actor's relations. If one state variable is changed, this affects how other state variables are evaluated whenever they are dependent.

4 MILITARY DOMAIN RULES

When a behavior is triggered, an actor response is executed and the actor's state variables are changed. The changes that occur on the actor's state variables are defined using a rule-based engine (Ray 2003). The rule-based engine uses military domain rules that are defined with the help of a subject matter expert and are stored in text files. These rules are set up as *if-then* rules which means that the engine checks if certain parameters have reached a specific condition. It then executes the necessary changes on the target parameters as shown in Table 1. In this particular example the engine checks if parameter A is greater than 0, etc. If this is the case it sets the value of parameter B to 2 and G to 1. This example basically states that if an actor's overall capability is high, then we can improve that actor's living condition and geographical dominance in the region.

Table 1: Example of a logical rule associated with an actor response.

If	**Then**	**Description**
A $>0 \wedge$ C $>2 \wedge$ D $>1 \wedge$ G >1 $\wedge$ I $>0 \wedge$ J $>1 \wedge$ K $>1 \wedge$ L >1 $\wedge$ N $>0 \wedge$ O >1	Set(B,2) $\wedge$ Set(G,1)	Update living conditions and geographical dominance of target.

5 ANALYSIS CLASSES

The main result from the simulator is the most efficient and effective plan based on thousands of simulation runs. Although this result might be enough for specific purposes, in addition an analyst might also like to check if there are other closely related plans. For this and other purposes we have developed analysis classes.

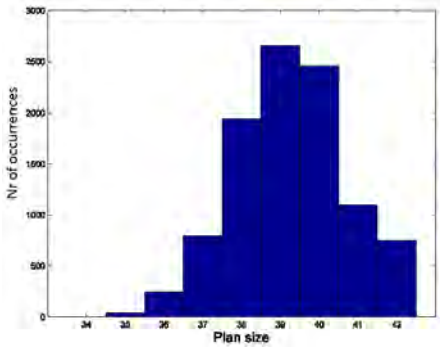
The analysis classes use the simulator as a source of input in order to perform different calculations. For example, one analysis class performs so called "complementing simulations". What this means is that the best plan is selected and for each tree level in the plan, one action is switched with another alternative. This process creates a new plan which is simulated and stored for later comparison. When the action on the last tree level is switched and the newly created plan is simulated, the process is finished and the accumulated data can be used for sensitivity analysis.

6 RESULTS

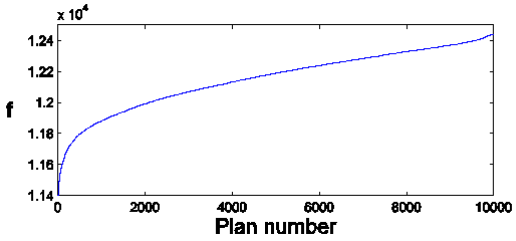
The simulator produced the first 10 000 most efficient and effective alternative plans based on a test case involving a complex military operation (total 2.1×10^{23} plan alternatives). The plans contained on average around 39 actions as shown in Figure 8(a).

The 10 000 plans were sorted based on their f-value in ascending order as shown in Figure 8(b). The best plans were obtained after only a few hundred plan simulations. Figure 8(c) shows the f-values based on the plan sizes. Here the trend showed that on average, the more actions were in a plan, the higher the f-value became, on contrary, the h-value decreased as the plan size increased as shown in Figure 8(d). This should not be accepted in general because we could have a shorter plan with much more efficient activities that would allow us to reach the end state faster. The h-values never reached near 0, which meant that we did not reach the end state in a satisfying way, but we managed to push the scenario towards the goal. The emphasize was on the h-value, hence we decided to weight it with a factor of 80 based on trial and error tests. The goal of the tests was to find the lowest distance values based on a specific weight as shown in Table 2.

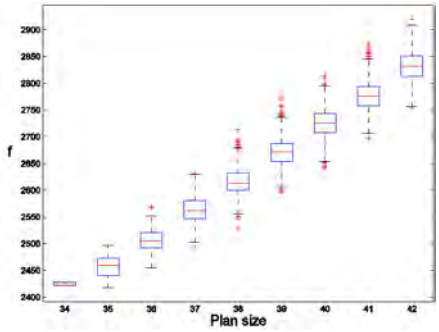
In Table 2, the first column indicates the tested weight, the second column indicates the minimum h-value found among all plans (total 10 000). The third column indicates the h-value for the minimum f-value found among all plans. The fourth column shows the time it took to go through all 10 000 plans based on the current weight. The fifth column shows the minimum g-value found among all plans and the



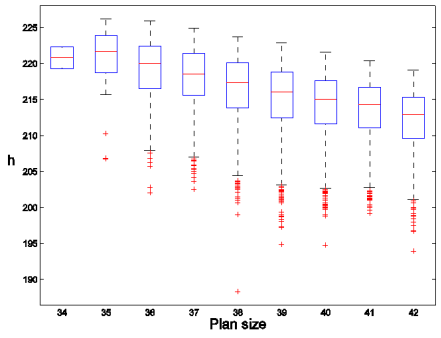
(a) The distribution of plan sizes for the first 10 000 plans.



(b) f-values for 10 000 plans.



(c) The f-value for average plan sizes.



(d) The h-value for average plan sizes.

Figure 8: Four figures showing different views of the simulation output data.

Table 2: Trial and error tests, weights and corresponding distance values.

Weight	**Min.** h	h **for min.** f	**Time (minutes)**	**Min.** g	g **for min.** f
60	794	794	8315	2267	2305
80	792.3	793.1	3994	2196.5	2422
90	795.3	795.3	1296	2199.8	2445.7
100	793.5	794	1131	2212.5	2406.5

sixth column shows the g-value for the minimum f-value found among all plans. Based on this information we chose 80 as weight due to the fact that the minimum h-value was low enough, as well as some of the other values for this specific military operation. On average, the time it took to generate, simulate and evaluate one plan alternative was 24 seconds.

7 CONCLUSIONS

In this paper we have shown the design and implementation of a simulator for simulation of military operative plans. There are different lessons that can be learned from this work. From our experience the best solution for designing such a system is to clearly separate scenario data from actors' behavior rules as much as possible. This is why we created an XML-model to contain the military plan and at the same time created a separate behavior engine for processing military domain rules, created by subject matter experts.

Implementing the simulator gives rise to certain challenges which need to be addressed. The main challenge from a technical point of view is the performance issue. In order to gain a much greater

performance boost we need to introduce parallelization throughout the code. The Monte Carlo method is an example where each iteration could be parallelized for better execution efficiency.

ACKNOWLEDGMENTS

This work was supported by the FOI research project "Real-Time Simulation Supporting Effects-Based Planning", which is funded by the R&D programme of the Swedish Armed Forces.

REFERENCES

Allen, T. T. 2011. *Introduction to Discrete Event Simulation and Agent-based Modeling: Voting Systems, Health Care, Military, and Manufacturing*. Berlin: Springer.

Body, H., and C. Marston. 2010. "The Peace Support Operations Model: Origins, Development, Philosophy and Use". *Journal of Defense Modeling and Simulation: Applications, Methodology, Technology* 8 (2): 69–77.

Dijkstra, E. 1959. "A Note on Two Problems in Connexion with Graphs". *Numerische Mathematik* 1 (1): 269–271.

Hart, P., N. J. Nilsson, and B. Raphael. 1968. "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". *IEEE Transactions on Systems Science and Cybernetics* 4 (2): 100–107.

Hinrichs, T. R., K. D. Forbus, J. Kleer, S. Yoon, E. Jones, R. Hyland, and J. Wilson. 2011. "Hybrid Qualitative Simulation of Military Operations". In *Proceedings of the Twenty-Third Conference on Innovative Applications of Artificial Intelligence*, 1655–1661.

Horne, G., and T. Meyer. 2010. "Data Farming and Defense Applications". In *Proceedings of the 2010 MODSIM World Conference and Expo*, 74–82. Hampton, Virginia.

Hunerwadel, J. P. 2006. "The Effects-based Approach to Operations: Questions and answers". *Air & Space Power Journal* 20:53–62.

Ligeza, A. 2001. *Logical Foundations for Rule-Based Systems*. Berlin: Springer.

Luotsinen, L. J., and E. Sjöberg. 2012. "The Modeling and Analysis of Computer Generated Forces Representing Groups and Organizations in Military Conflict Zones". In *Proceedings of the NATO Symposium on Transforming Defense through Modeling and Simulation Opportunities and Challenges*.

Pourret, O., P. Naim, and B. Marcot. 2008. *Bayesian Networks: A Practical Guide to Applications*. Chichester: John Wiley & Sons.

Ray, E. T. 2003. *Learning XML*. Second ed. Sebastopol, California: O'Reilly Media.

Schubert, J., F. Moradi, H. Asadi, P. Hörling, and E. Sjöberg. 2010. "Simulation-based Decision Support for Effects-based Planning". In *Proceedings of the IEEE International Conference on Systems Man and Cybernetics*, 636–645. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Surdu, J. R., and K. Kittka. 2008. "The Deep Green Concept". In *Proceedings of the Spring Simulation Multiconference, Military Modeling and Simulation Symposium*, 623–631.

AUTHOR BIOGRAPHIES

HIRAD ASADI received the M.Sc. degree in computer science from the Royal Institute of Technology, Stockholm, in 2009. He has worked as a software consultant for SAAB Combitech after graduation, before joining the Swedish Defence Research Agency, FOI. His research interests are modeling and simulation within the military domain. His email address is hirad.asadi@foi.se and his web page is http://www.foi.se/fusion.

JOHAN SCHUBERT is a Deputy Research Director with the Division of Information and Aeronautical Systems at the Swedish Defence Research Agency, FOI. He received an M.Sc. in Engineering Physics in 1986 and his Ph.D. in Computer Science in 1994, both from the Royal Institute of Technology, Stockholm. He has conducted research in Information Fusion and Decision Support for 26 years and published 16

journal articles, four book chapters and 40 conference papers. He was the technical program chair of the Seventh International Conference on Information Fusion (FUSION 2004), co-editor of the conference proceedings and guest co-editor of a double special issue of the journal Information Fusion on FUSION 2004. He is a board member of the Belief functions and Applications Society and member of the editorial board of the Information Fusion Journal. His current research interests include theoretical and applied aspects of Soft Computing, Neural Networks, Evolutionary Algorithms, Modeling and Simulation, Data Farming, Management of Uncertainty and Dempster-Shafer Theory, military applications of high-level Information Fusion and Artificial Intelligence for Situation and Threat Assessment and its use in Decision Support Systems. His email address is johan.schubert@foi.se and his web page is http://www.foi.se/fusion.