Knowledge Representation Supporting
Multiple Reasoning Methods for Simulated Operators

Ruibiao J. Guo
Brad Cain
Pierre Meunier
DRDC Toronto
1133 Sheppard Avenue West
PO Box 2000
Toronto, Ontario M3M 3B9
jaff.guo@drdc-rddc.gc.ca, brad.cain@drdc-rddc.gc.ca, pierre.meunier@drdc-rddc.gc.ca

Keywords:
Knowledge Representation, Simulated Operators for Networks, Reasoning in IPME Task Nodes, Multiple Reasoning Methods, Computer-Generated Forces

ABSTRACT: Modeling and simulation of human behaviours plays a significant role in Computer Generated Forces (CGF). There is a growing recognition that the current models of military operators are insufficient in many simulation-based applications. DRDC Toronto is considering an architecture, SIMulated Operators for Networks (SIMON), comprising a computerized toolset for creating models of military personnel that are suitable for use in distributed simulations over a wide range of applications. This paper proposes a language, LAMP (Language of Agents for Modeling Performance), to represent and organize knowledge for tasks to extend the reasoning capabilities of simple task network modeling environments such as the Integrated Performance Modeling Environment (IPME).

1. Introduction

Modeling and simulation plays a significant and expanding role in force development, acquisition, training, and mission rehearsal. In some, physics-based models of systems or the environment will be prominent; in others, operator models will be dominant. Ideally, we would be working with a suitable balance of physics and human models in order to produce the desired results, but in all cases, some measure of each type of model is required. As the application area becomes increasingly complex and unstructured, there will be a greater need for more accurate representation of the operator.

There is a growing recognition that the current models of military operators are insufficient in many applications as noted in the US NRC (Pew and Mavor, 1998), NATO RTO SAS 017 (Dompke et al., 2001), and TTCP HUM AG19 (Hawkins et al., 2003) reports on human behaviour representation and computer generated forces in military modeling and simulation. Most AI CGF use simple forward chaining that does not support diagnostic, inductive, abductive or uncertainty reasoning (Pew and Mavor, 1998; Endsley, 1995).

DRDC Toronto is considering an architecture, SIMulated Operators for Networks (SIMON), as shown in Figure 1. SIMON will comprise a computerized toolset for creating models of military personnel for a range of applications to support human behaviour simulation for the Canadian Forces. The intention is to introduce a modeling philosophy to the toolsets currently used within the Canadian Forces that will allow analysts to extend operator models developed in the early stages of human factors engineering analyses to form the basis for training-simulation operator models with the goal to capitalize on the investment made in the knowledge representation in the HFE analysis. SIMON-based operators will be implemented as independent, networked operators rather than embedded entities within a monolithic simulation to promote their reuse and extension to other simulations whether they be for analysis or for training (Cain and Kwantes, 2004). SIMON is not an attempt to develop a Unified Theory of Cognition, rather it is a pragmatic attempt to introduce psychologically plausible models into engineering approaches to human behavior representation to improve that representation when and as required, capitalizing on our HFE tool investment.

The proposed SIMON architecture comprises a number toolboxes including: perception and operator state, performance, cognitive, emotional and diagnostic. Users will select toolboxes and configure their virtual operators as required, choosing from a library of component models in a modular fashion. The structure in Figure 1 integrates existing IPME task network...
modelling approach (Micro Analysis & Design, 2001) and other modules developed or implemented by DRDC Toronto, such as a memory model, motor models and performance moderator functions. SIMON will interact with external synthetic or simulated environments (controlling a physical helicopter simulator is planned).

**Figure 1.** SIMON (SIMulated Operators for Networks)

IPME is the principal simulation engine within Simon. The main features of IPME that we use in Simon are the task network drawing tool for representing procedural knowledge as well as task data, the crew model for assigning operator attributes, and the performance shaping function mechanism that is used to moderate behaviour through component models of operator states that may evolve with the simulation.

Figure 2 shows a typical task network in IPME. The task network model allows users to easily describe the processes used by a human operator to perform an activity. It also addresses the design parameters of the workspace in which the processes must be performed (such as the location of work surfaces, controls and displays, and other objects that the operator must manipulate), and the use of those design parameters to calculate times and accuracies of the processes in the activity. Task networks in IPME support hierarchical decomposition of processes and have embedded models to govern workload.

One thing that IPME does not do conveniently is to provide a mechanism for creating models of higher order cognition, such as human-like memory or reasoning. We believe that the combination of techniques such as Hierarchical Task Analysis (Annett et al. 1971), the simulation capabilities of IPME, and the introduction of selected models from both cognitive psychology and artificial intelligence communities will provide Simon with the necessary tools to build useful operator models for military simulations.

**Figure 2.** A Task Network in IPME.

The reasoning module in Simon should support the discrete inference associated closely with tasks in IPME, meet the requirements of real-time environment related to task network simulation, and support multiple reasoning methods. While popular cognitive architectures, such as ACT-R (Anderson et al., 2004) and Soar (Laird et al., 1987), offer a great deal of flexibility, they also come with restrictions. For instance, ACT-R is somewhat hampered in real-time applications due to constraints introduced by LISP (Best and Lebiere, 2005). Soar is not appropriate for low-level sensory simulations and use in non-deterministic domains (Soar Technology, 2005).

In this article, we introduce a language, LAMP (Language of Agents for Modeling Performance), designed to sustain SIMON’s special requirements for task-node-oriented, discrete, and real-time reasoning. The next section is an overview of the language, and the coming sections depict the details, examples, and the way to interact with the simulation engine IPME.

### 2. Overview of LAMP

The design goals of LAMP include (1) developing a flexible modular approach to reasoning that extends the primitive reasoning capabilities of current discrete event simulators intended for HBR, (2) providing the user with a mechanism to support multiple reasoning methods in discrete event simulation engines running in real-time, and (3) demonstrating this reasoning capability in a simple applied problem. Figure 3 shows the overview of the language structure that consists of two kinds of components: aspects that are small reasoning units related to task nodes in IPME and reasoning engines that help infer conclusions based on...
task-related data and aspects; definitions for terms used in Figure 3 are characterized in the coming sections.

![Figure 3. Overview of LAMP](image)

The proposed language LAMP is being implemented in C++ under Linux. An application for Air Traffic Control conflict resolution (Loft et al., 2004) is also being developed. The details of implementation will be reported in the future.

### 3. Basic Elements

In this section, we first review and define some terminology and basic components used in the representation language. We will use an Air Traffic Control scenario as a specific example of the general concept.

LAMP uses labels, numbers, time and lists as basic data types.

Sets and quantifiers are two concepts frequently used in logic. We employ angle brackets “<” and “>” with enclosed members to represent a set and “@” to designate a containing relationship of sets. We also use quantifiers to indicate the quantity of a class that has a property. For example, the following expression, \( \text{all } \text{aircraftX}@\text{aircraftList} \{ \text{aircraftX->speed < 630} \} \), represents a quantifier relationship testing: if the speed variable of each aircraft in aircraftList is less than 630 miles per hour, the relationship is true.

Knowledge elements in LAMP consist of entity relations, precepts and aspects. An entity relation represents an object with attributes that might be a declaration, predicate or relationship between entity attributes. Precepts deal with procedures or rules. An aspect in LAMP is a knowledge unit that sustains reasoning functions associated with task nodes in IPME. The support relationships of these elements are entity relations \( \Rightarrow \) precepts \( \Rightarrow \) aspects \( \Leftrightarrow \) IPME task networks. In this section, we describe entity relations and precepts. Aspects are addressed in the coming sections.

#### 3.1 Entity Relations

Entity relations and precepts are two basic elements in LAMP, representing primitive declarative and procedural knowledge respectively.

An entity relation describes an object with attributes that might be affirmed or denied with logic TRUE or FALSE values depending on reasoning methods. For first order logic reasoning, each entity relation associates with a TRUE/FALSE value. In fuzzy logic based reasoning, however, each entity relation requires only instantiations of attributes, fuzzy members and degrees of memberships. Each entity relation consists of a schema that defines a declaration, and a collection of instances that are related to facts.

An entity relation schema is an outline of an entity relation, subsuming a name and a list of parameters enclosed within a pair of brackets. The syntax of an entity relation schema is as follows:

\[
[\text{relationName}, \text{type1:parameter1}, \{\text{constraintList1}\}, \text{default1}; \ldots \text{typei:parameteri}, \{\text{constraintListi}\}, \text{defaulti}; ...],
\]

where \( \text{relationName} \) is an identifier, \( \text{typei} \) is the data type of the \( i \)-th parameter with \( \text{parameteri} \) as name, \( \text{constraintListii} \) is a collection of \( \text{constraints} \), and \( \text{defaulti} \) is the default value of \( \text{parameteri} \).

Constraints are restrictions to values of parameters in entity relations. Each constraint contains an operator and a list of constants. For example, ‘\( \text{oneOf "headOn", "catchUp", "crossing"} \)’ is a constraint, where ‘\( \text{oneOf} \)’ is an operator and ‘\( \text{"headOn"}, \text{"catchUp"}, \text{"crossing"} \)’ is a constant list. This constraint means that the value of a variable with the constraint can only be either ‘\( \text{"headOn"} \)’ or ‘\( \text{"catchUp"} \)’ or ‘\( \text{"crossing"} \)’.

The following are examples of entity relations for different kinds of knowledge, including first order logic, fuzzy logic and probability-based knowledge.
First order logic based entity relations are used in many AI applications. In the air traffic conflict resolution area, a conflict type is bound with a name, an identifier and an aircraft type list. In a special case, the aircraft types might be “B747”, “B767”, and “A320”. The following entity relation schema depicts this relation:

```
[conflictType, label:typeName, {oneOf <"headOn", "catchUp", "crossing"}>, "headOn";
label:id, {maxLength < 1024 >}, "unknownId";
list:acTypeList, {memberOf<"B747", "CF1175", "A320", "A380">}, NULL].
```

An instance of this entity relation is as follows, `[conflictType, "headOn", "CF1175", <"B747","A320">]`, where, “headOn” is the name of conflict type; “CF1175” is the conflict identifier; and `<"B747","A320">` is the list of aircraft types involved in this conflict.

We could use a fuzzy relation to represent an aircraft’s taking off time, including an a/c name, fuzzy member (“past”, “recent” or “future”), and a degree of membership with a value between 0.0 and 1.0. One of the schemas for the context is

```
[takingOffTime,
label:callSign, {maxLength < 1024 >}, "unnamed";
labeled:offTime, {oneOf<"past", "recent", "future"}>, "recent";
number:degreeOfMembership, {rangeBoth <0.0,1.0>}, 0.0].
```

A possible context is that “AC354 took off recently”. An instantiated fact for this context is `[takingOffTime, "AC354", "recent", 0.7]`, where the number 0.7 indicates the degree that the time of taking off is close to the grade “recent”.

Probability based entity relations are another kind of uncertainty relations. For the following context: “the probability that a flight performs well is x, given that turbulence occurs”, its schema might be

```
[performProb, label:callSign, {}, "";
label: hypothesisEvent, {}, "";
list: evidenceList, {}, <>
number: probability, {rangeBoth <0.0,1.0>}, 0.0].
```

An instance for this probability relation is `[performProb,"B67", "performWell", <"turbulenceOccurs">, 0.745]`

### 3.2 Precepts

A precept in LAMP, comparable to a rule in rule-based systems, is a minimum reasoning unit that derives conclusions from a group of premises or conditions. The syntax of precept is prescribed as the form given below:

```
PreceptType PreceptName{condition list => conclusion list}
```

A precept consists of a PreceptType that might be first order logic, fuzzy, or probability, a PreceptName, a group of conditions and a set of conclusions. If all conditions meet the requirements of type-related reasoning engines (in the case of first order logic, all conditions are true; and for fuzzy logic, all conditions are instantiated), the conclusions will be derived.

A condition in a precept is a relation or expression that might be one of the following categories: an entity relation, an expression, a variable assignment, or an expression relation testing.

A conclusion in a precept is a result derived from a collection of conditions that is one of the following types:

1) an entity relation instantiation
2) entity relation/variable manipulations, such as assigning, storing, updating & deleting
3) external communications, for example, sending conclusions or commands to behavior simulation engines
4) native function activation, e.g. executing commands or calling functions
5) cognitive activation, for instance, further reasoning.

The following simple examples show first order logic and fuzzy precepts respectively.

If the current conflict type between `aircraftA` and `aircraftB` is “headOn”, the level above is available, and `aircraftA`’s climbing performance is good, we might conclude that, in order to solve this conflict, `aircraftA` climbs 1000 feet. The corresponding precept is stated as follows:

```
FOLPrecept PHOa {
[conflictType, “headOn”] &&
[levelAboveAvailable, aircraftA, “yes”] &&
[performance, aircraftA, “good”] &&
[feasible, “climb”, < aircraftA, 1000>, “yes”]
=>
climb (aircraftA, 1000) }.
```

We can also describe fuzzy rules with precept syntax. If the current conflict type is “headOn” and available level above; `aircraftA` is not taking off recently; `aircraftB`, with good climbing performance, is far from destination; then we conclude that `aircraftB` climbs up
with 1000 feet. The following precept characterizes this context.

FuzzyPrecept PHOb {
    [conflictType, "headOn", dom1] &&
    not[takeoffTime, aircraftA, “recent”, dom5] &&
    [performance, aircraftB, “good”, dom8] &&
    [feasible, “climb”, <aircraftB, 1000>, “yes”, dom9]
    =>
    [climb, <aircraftB, 1000>, dom10] }.

In the process of reasoning, if all conditions can be instantiated with data acquired from IPME or previous reasoning, the conclusion in this precept is derived.

4. Aspects – Knowledge Units for Reasoning

Aspects in LAMP are knowledge units related to reasoning associated with task nodes in IPME. Each aspect embodies attributes, interaction interfaces and descriptions of entity items or relations, facts, precepts or procedures. With the focus to reasoning, aspects support multiple strategies, inherence and reuse, consistency, autonomy and learning.

A schema in the following format defines an aspect:

```
AspectType AspectName (<taskList>, <importList>):
    <parentList> {
        MetaSet {...};
        SenseSet {...};
        RelationSet {...};
        FactSet {...};
        PreceptSet {...};
        MethodSet {...};
    }.
```

In this schema, AspectType might be first order logic, fuzzy logic or probability depending on requirements of reasoning in IPME task nodes. Actually, the AspectType slot is flexible enough to support more reasoning or knowledge manipulation methods, such as neural-fuzzy reasoning. However, for each AspectType, there should be corresponding engines or modules in the LAMP system to sustain relevant reasoning or knowledge utilization.

AspectName is a unique identifier in the system databases.

The taskList links this aspect to task nodes in IPME task networks. The association relationships between aspect nodes and task nodes in IPME might be one-to-one, one-to-many or many-to-many relations, i.e. a task node in IPME may associate with one or more aspects nodes, and an aspect node could be used by one or more task nodes.

The importList provides a facility to integrate multiple files together, with which a large aspect can be developed in parallel by multiple users and stored in different locations or machines before pre-compiling and running.

The parentList is used for multiple inheritances similar to that in object-oriented methodology. With this mechanism, a complex reasoning might be organized into a hierarchy of aspects, each of which handles sub-goals of reasoning. The hierarchy of aspects can be activated as a whole by certain tasks in IPME.

MetaSet contains meta-attributes for global uses, for instance, indicators for extendable application areas or statistical information (such as frequency of use) of this node. In future, users may employ search tools provided by LAMP system to find aspects developed previously in relevant application areas for reuse or referring to reasoning units. In addition, if users declare some statistical information related to this aspect, the information could be used to evaluate the usability of this aspect, and, further, to refine and improve reasoning performance.

The SenseSet describes external and internal variables used by this aspect. External variables hold data coming from the external environment through the interfaces between the LAMP system and the simulation engine IPME. Internal variables contain derived data from external variables through procedures or algorithms defined in this aspect. For example, a reasoning engine acquires current temperature in degrees from external environment through IPME, but the measure of current temperature in reasoning is in fuzzy grades. As a result, two variables are used to represent both external degrees and internal fuzzy grades. In implementation, the LAMP system interacts with the simulation engine IPME through HLA compatible network communications or native interfaces to attain current situation data and then, if necessary, converts them into internal forms for reasoning.

RelationSet encompasses declarations of entity items or relations in this aspect. An entity item or relation consists of an item name and a list of parameter descriptions, each of which contains a parameter name, data type, constraints and default values. In the process of reasoning, all facts or patterns have to meet the restrictions of data types and constraints described in
the declarations. This helps maintain data consistency and reduce errors in reasoning processes.

*FactSet* is a collection of constants of entity items or relations defined in the *RelationSet*. This is the declarative memory of this aspect, storing facts or relations. In reasoning processes, these facts can be used directly by rules in the *PreceptSet*. Facts can be put into this section when the aspect is developed. Furthermore, with reasoning processes, new facts can be added to this section based on commands or actions defined in the rules in *PreceptSet* section.

*PreceptSet* includes a group of precepts or rules supporting unit reasoning. Each precept or rule contains conditions and conclusions or consequences. When an IPME task node invokes this aspect, or related data or events in external environment changes, a relevant reasoning engine activates rules in *PreceptSet*, verifies conditions in rules, derives conclusions and notifies the simulation engine the conclusions. Moreover, the simulation engine acts based on the derived conclusions.

*MethodSet*, another optional section in an aspect, comprises all procedures or algorithms involved in this node that can be called by other sections or children of this aspect. As an example, if the distance data from external environment is in miles, and the required distance for reasoning in this aspect is in fuzzy grades such as “far” and “near”, it is necessary to develop an algorithm to convert the miles to fuzzy grades used in reasoning.

In summary, an aspect is a reasoning unit related to IPME task nodes, which supports multiple strategies, inherence and reuse, consistency, autonomy and learning. The *AspectType* slot in an aspect, working together with a relevant reasoning engine, deals with different strategies for reasoning or knowledge manipulation. Another slot, *parentList*, contains information about its parents that provides multiple inherence mechanism. Using such inherence, users can decompose complex reasoning into subgoal hierarchies. In LAMP system, an aspect can be reused for different task nodes, task networks, and even different projects in IPME. Also, the meta-attribute section in an aspect can help users hold meta-information about this aspect for extended applications or advanced performance analysis and learning. The definitions and declarations of entity items in *RelationSet* keep data consistency and reduce errors in reasoning processes. The *SenseSet*, *FactSet*, *PreceptSet* and *MethodSet* in an aspect support autonomy to a certain degree with the capabilities for perception, decision-making and actions. While the *SenseSet* perceives data from external environment through the simulation engine IPME, the *FactSet* and the *PreceptSet* are able to derive conclusions and assist decision-making. The *MethodSet* supports both related actions and internal procedures in reasoning processes.

The following is an example of aspect with fuzzy logic used to adjust speed of armed vehicles when it is snowing. This aspect obtains snowing data, converts them into fuzzy grades and derives conclusions for speed adjustment.

```
FuzzyAspect inferSpeed (<"HandleSnowing">, <>):
  "reasoningRoot" { }
  MetaSet { }
    list usedTo = <"Armed Vehicles", "Armed Robots">;
  SenseSet { }
    extern long snowingDensity;
    extern long speed;
  RelationSet { }
    [snowing, label: grade, {oneOf="heavy", "light"}, "light";
      number:snowingDom, {rangeBoth<0.0,1.0>,[0,0]}];
    [speed, label: grade, {oneOf="fast", "slow"}, "fast";
      label:vname, {}, "";number:speedDom, {rangeBoth<0.0,1.0>,0.0}];
    [increaseSpeed, label: vname, {}, "";number:kmPerHour, {}, 5; number:speedDomi, {rangeBoth<0.0,1.0>,0.0}];
    [reduceSpeed, label: vname, {}, "";number:kmPerHour, {}, 5; number:speedDomr, {rangeBoth<0.0,1.0>,0.0}]
  PreceptSet { }
    FuzzyPrecept snowSpeed1 { }
      [snowing, "heavy", snowingDom11] &&
        [speed, vname, "fast", speedDom12] =>
        [reduceSpeed, vname, 5, speedDom13]
    } FuzzyPrecept snowSpeed2 { }
      [snowing, "light", snowingDom21] &&
        [speed, vname, "slow", speedDom22] =>
        [increaseSpeed, vname, 5, speedDom23]
    } MethodSet { }
    number getSnowDom(label member, number snowingDensity) { }
      if(member="heavy"){
        if (snowingDensity<=1) return 0.0;
        else return (snowingDensity -1)%2;
      }
      else if (member="light"){
        if (snowingDensity >= 1) return 0;
        if(snowingDensity <= 0) return 0;
        if(snowingDensity <= 0.5) return 2*snowingDensity;
        if (snowingDensity > 0.5) return (1-snowingDensity) *2;
      }
    number getSpeedDom(label member, number speed) { … }
  }.
```
This aspect node, with type *FuzzyAspect* and name *inferSpeed*, relates to the task node “HandleSnowing.” Its parent node is “reasoningRoot”. The *MetaSet* contains the application areas that this node can be used for, e.g. “Armed Vehicles” and “Armed Robots”. The *SenseSet* section indicates two situation variables, “snowingDensity” and “speed”, used in reasoning. In the *RelationSet* section, there are four entity items defined: “snowing”, “speed”, “increaseSpeed” and “reduceSpeed”, which will be used in precepts. The *PreceptSet* defines two fuzzy rules for increasing or reducing vehicle speed based on current snowing and vehicle speed situation. The *MethodSet* describes procedures that are used to convert situation data into fuzzy degrees of membership used in the reasoning process.

5. Applications and Interaction with Simulation Engines

5.1 Applications

An application in IPME consists of environment models, crew models, task network models and performance shaping functions. IPME users are able to develop task-related aspects for reasoning used in their task network models. Figure 4 is a portion of an IPME application for air traffic control conflict resolution. In the task network, there are four task nodes: “DetectConflicts”, “HandleHeadOn”, “HandleCrossing” and “HandleCatchUp.” These task nodes require reasoning function for deriving necessary manoeuvres related to current situation before they can take actions to resolve conflicts.

The following are details of one of aspects, “InferHeadOnManoeuvre”, in Figure 4. The variables and entity relations without definitions in this node can be inherited directly from its parent node, “AspectRoot”, where some common variables and entity relations are declared. This aspect consists of two first order logic precepts for aircrafts climbing and turning respectively.

```
FOLAspect InferHeadOnManoeuvre
(<"HandleHeadOn">, <>): <"AspectRoot"> 

{ 

MetaSet 
{ 

list usedTo = <" ATC Conflict Resolution"} 

SenseSet 
{ 

extern long distance} 

RelationSet 
{ 

[headOnDistance, label:, {oneOf <"short", "medium", "long"}]} 

FactSet 
{ 

PreceptSet 
{ 

FOLPrecept PHO1 
{ 

[conflictType, “headOn”] &&
[conflictTime <= 10] &&
[turbulence, “no”] &&
[levelAboveAvailable, callSigns[i], “yes”] &&
not [takeoffTime, callSigns[i], “recent”] &&
[performance, callSigns[i], “ok”]
=>
climb(callSigns[i], 1000) } 

FOLPrecept PHO2 
{ 

[conflictTime, recent, ] &&
[turbulence, “yes”] &&
[levelAboveAvailable, “no”] &&
[takeoffTime, callSigns[0], “recent”] &&
[performance, callSigns[0], “inadequate”] &&
[destination, callSigns[1], “far”] &&
[headOnDistance, “short”] &&
[feasible, “climb”, <callSigns[1], “left”, 15>, “yes”]
=>
turn(callSigns[1], “left”, 15) 

}.
```

5.2 Interaction with Simulation Engines

Figure 5 shows the interaction between this system and the simulation engine, IPME. While application modellers develop IPME task networks, they also build task-related aspects for reasoning as required. In the simulation process of task networks, IPME is working as a server, and the reasoning module as a client. The reasoning module requests timing reasoning services from IPME. When a task node needs to call the reasoning module to drive conclusions, the IPME server notifies the reasoning module when it is okay for them to proceed. This is achieved by inserting events into the IPME simulation. When the registered services are activated at the time scheduled by IPME, the reasoning module requests relevant situation data, does
inference, and sends back the derived results. In such simulation environment, it is significant for the reasoning module to synchronize activities with IPME’s clock, events, and synchronization mechanisms.

6. Conclusions

The LAMP reasoning language described in this paper is an approach to knowledge representation intended to extend the current reasoning capability of discrete event simulations such as IPME in order to make task networks more suitable for use in representing human behavior. LAMP attempts to integrate features from task nodes in IPME, real-time and multiple methods including first-order logic, fuzzy logic, and probabilistic reasoning. The LAMP structure may integrate hybrid-reasoning approaches, but the current focuses are still on independent reasoning engines. At this point, the work is concentrated on developing the reasoning engines themselves; future work will incorporate methods to implement different reasoning styles such that the analyst can model decision-making based on psychologically plausible styles and reflecting operator biases.

LAMP is intended to be a component of a larger project to model military operators as networked agents for use in analysis and training applications. LAMP works with other modular components to bring together component models of human behavior and performance to better represent human behavior modeling in various military simulations.

The SIMON project is still in the planning and architecture phase. As a preliminary proposal, many issues in this reasoning representation still need to be verified or adjusted in the development progress of the project SIMON.

Acknowledgments: This research is supported by DRDC Toronto through an NSERC grant. We appreciate the contributions from Dr. Peter Kwantes in DRDC Toronto for helping us shape the direction and content of this research. We thank the anonymous reviewers very much for their precious suggestions and comments.

7. References


Author Biographies

RUIBIAO J. GUO is a visiting fellow supported by DRDC Toronto through a NSERC grant. His research interests include AI, reasoning, human behavior modeling and intelligent tutoring systems. He has a B.Eng. and M.Eng. in Computer Science from the Hefei University of Technology, China, and a Ph.D. in Computer Science from the University of Montreal, Canada.

BRAD CAIN is a Defence Scientist and Professional Engineer at DRDC Toronto, having a B.A.Sc. and an M.A.Sc. in Mechanical Engineering from the University of Toronto. His research interests include human behavior and performance modelling and he is currently the Project Manager for a human behaviour representation project "Simulated Operator for Networks" that is exploring methods for creating cost-effective, reusable operator models for training simulations.

PIERRE MEUNIER is the leader of the Human Modelling Group at Defence R&D Canada - Toronto. He has a B.Eng (mechanical) from McGill University and an MSc in Industrial and Systems Engineering (Human Factors) from Virginia Tech.