Proceedings of the 2010 Winter Simulation Conference
B. Johansson, S. Jain, J. Montoya-Torres, J. Hugan, and E. Yücesan, eds.

IMAGE-SCENARIZATION: A COMPUTER-AIDED APPROACH FOR AGENT-BASED ANALYSIS AND DESIGN

Michel Lizotte

Defence R&D Canada
2459, Pie-XI Blvd North,
Québec (Québec) G3J 1X5 CANADA

François Rioux

LTI Software and Engineering
825 Boul. Lebourgneuf, Bureau 204,
Québec (Québec) G2J 0B9 CANADA

ABSTRACT

Agent-based modeling has been of interest to researchers for some time now. Some research has focused on the analysis and design of such software, but none has truly addressed the need for automated assistance in creating agent-based simulators from initial problem comprehension. This paper proposes an approach addressing the gap and supporting the spiral process of generating an agent-based simulator. In particular, this approach enables the incremental and iterative representation of a problem and its translation into an executable model. Initially using an unconstrained ontology, the designer draws conceptual graphs representing the problem. Progressively, graph elements are linked hierarchically under concepts that are part of a predefined generic Scenarization Vocabulary (i.e., agent, patient, behaviour, attribute, parameter, variable …). This Scenarization semantic defines roles in the simulation. This approach is part of a broader research effort known as IMAGE that develops a toolset concept supporting collaborative understanding of complex situations.

1 INTRODUCTION

A simulation based on current beliefs about a situation is either a mental act or an actual exercise performed by a human to better approach the future. For instance, to increase their grasp of a situation, the military use war-games to fake dynamics between friendly and enemy forces. The use of an executable model based on the current comprehension of a situation is a way to facilitate this action and better appreciate future possibilities.

A huge community of research is investigating agent-based modeling. Work by Macal and North (2006) discusses the analysis and design of such software. However, no study seems to truly address the need for automated assistance in creating agent-based simulators from initial problem comprehension. This paper proposes an approach addressing the gap, focusing on the lack of formalism and procedural maturity stressed by Macal and North (2006).

The work presented in this paper is performed under the IMAGE concept (Lizotte et al. 2008) that targets the collaboration of experts trying to reach a common understanding of a complex situation. The principles underlying the concept include: (1) Iterative understanding: a common understanding is reached through revision and sharing of successive representations of the situation; (2) Synergy of technologies: the common understanding is assisted by the synergy of tools for representation, scenarization, simulation and exploration; and (3) Humans in the loop: the common understanding is above all a human task supported by tools helping a person's comprehension method. This paper is about the 2nd principle. In particular, it focuses on the IMAGE-Scenarization approach (IMAGE-SCE).

Section 2 introduces the graphical notation used to define and scope the problem. The next two sections summarize the generic Scenarization Vocabulary and the Specification Schemas graphs used, as described in Section 5, to transform problem comprehension into an executable agent-based model. Using an example, Section 6 covers the spiral process of progressively creating a “validated” simulation.

2 GRAPHICAL NOTATION: REPRESENTING THE PROBLEM

The famous quote by Charles F. Kettering, a US electrical engineer and inventor (1876 - 1958), states: "A problem well stated is a problem half solved." This saying summarizes the intent of this part of the approach where the designer is assisted in posing the problem at hand.

A priori knowledge and gathering information from various sources is used to express a problem definition. This problem definition consists in drawing a set of conceptual graphs and, potentially, a custom vocabulary. Together, these graphs and this Comprehension Vocabulary constitute the comprehension model of the problem.

2.1 Comprehension Graphs

Figure 1 provides an example of the graphical notation of conceptual graph formalism (Sowa 1984) variation used in the approach. This example was produced using the CoGUI-IMAGE tool based on the CoGUI tool. The latter software is founded on the conceptual graph research efforts of Chein and Mugnier (2008) as well as on work by Genest (2010). Rectangles are concepts, while ellipses are conceptual relations. A concept has two parts, a type label before the colon and a marker after the colon. The type label represents the type of entity the concept refers to, while the marker refers either to the generic marker “*” or identifies actual individuals, also called referents. Arcs pointing toward or away from an ellipse mark arguments of the relation. The arrow orientation is devoid of semantics and serves only to ease graph reading. A concept can also be detailed using a nested graph as shown in Figure 1, where the concept “IED Event:*” is detailed with the graph “IED:*➔Explode:*.” IED means “Improvised Explosive Device.”

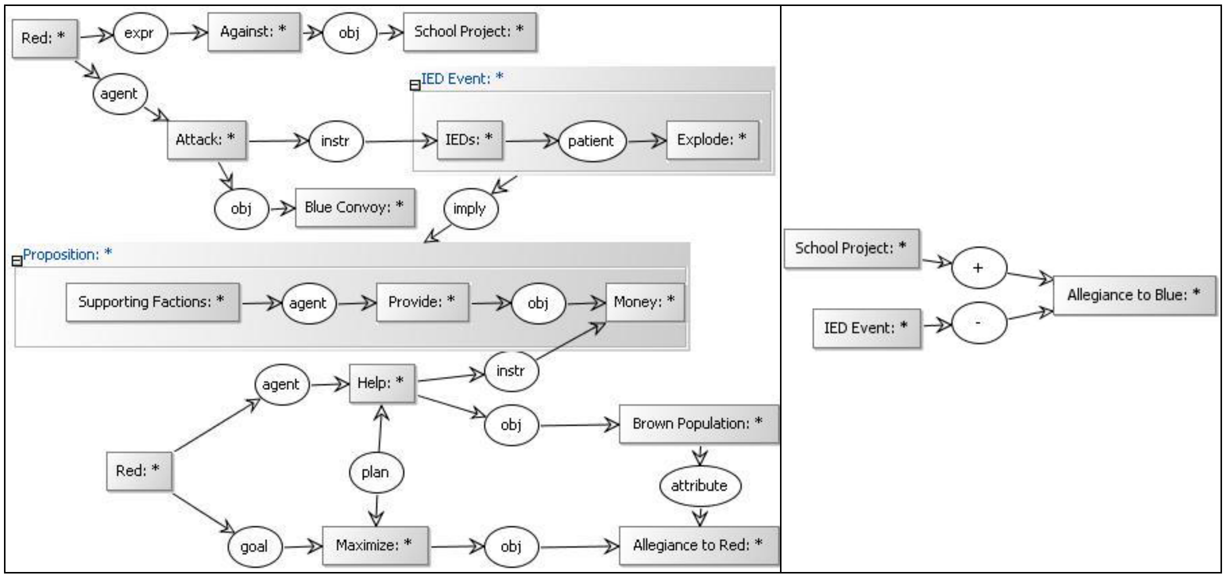

Figure 1: Graphs of a comprehension model

In the original conceptual graph notation, an arc pointing towards an ellipse marks the first argument of the relation, while one pointing away from an ellipse marks the last argument. If a relation has only one argument, the arrowhead is omitted. If a relation has more than two arguments, the arrowheads are replaced by integers 1,...,n. The original conceptual graph notation is described in Van Harmelen, Lifschitz, and Porter (2008). Conceptual Graphs are a system of logic based on the existential graphs of Charles Sanders Peirce and the semantic networks of artificial intelligence. They express meaning in a form that is logically precise, humanly readable and computationally tractable. They can be used as an intermediate language for translating computer-oriented formalisms to and from natural languages. They serve as a readable, but formal, design and specification language. They have been implemented in a variety of projects for information retrieval, database design, expert systems and natural language processing.

As the IMAGE-SCE was progressing, needs for lightening conceptual graphs were identified. As a consequence, the graph notation was extended to reduce the quantity of shapes and lines. For instance, the following extensions are used:

- A "Concept Group" is a rectangle including a set of N concepts (no relations) used to eliminate repetition of the same relation symbol (and arc). An arc between this rectangle and a relation actually means N relations (and arcs), one for each concept in the set.
- A "Set of referents" is a curly-bracket set expression used as a marker to reduce the number of concept symbols of the same type. Such a set composed of N members means N concepts. For instance, "Road Segment: {B,D,E}" actually means Road Segment: B" , "Road Segment: D" , and "Road Segment: E." A relation with such a symbol actually means N relations, as above.
- A "Vector of referents" is a label and interval bracket expression used as a marker to reduce the number of concept symbols of the same type. Such an interval from 1 to N means N concepts. For instance, "IED: x[1-3]" means "IED: x[1]," "IED: x[2]" and "IED: x[3]" .
- An "Array of referents" is a label and a dimension declaration in brackets used to reduce the number of concept symbols. Such an array means as many concepts as there are coordinates in the array. For instance, "Population: p[2 X 2]" means "Population: [1,1]," "Population: [1,2]," "Population: [2,1]," and "Population: [2,2]" .

2.2 Comprehension Vocabulary

In the context of IMAGE, which essentially uses the definitions of Chein and Mugnier (2008), a vocabulary is a simple ontology composed of a concept type set and a relation type set, also called hierarchies. Both sets are partially ordered by "is-a" (shown as an arrow) relations. In addition, a relation type has a signature specifying the arity of the relation and the "maximal" (hierarchically highest permitted) concept type of each argument. Using both, a textual tree and a graph layout, Figure 3 of the next section presents the concept types and the relation types of a vocabulary.

The comprehension vocabulary explicitly structures and labels concept types and relation terms used in the comprehension graphs. The creation of a this vocabulary is optional for the problem representation activity of the IMAGE-SCE approach. It is up to the designer to decide whether or not such vocabulary is needed. On the other hand, the Scenarization Vocabulary introduced in the following section is mandatory. Any concept or relation playing a role in the simulation will eventually have to be linked under the generic Scenarization Vocabulary.

3 SCENARIZATION VOCABULARY: IDENTIFYING SIMULATION OBJECTS

Starting from comprehension concepts and relations introduced above, the designer has to identify simulation objects. While the comprehension model leaves the user free to express any knowledge of interest, the Scenarization Model requires that all objects essential to simulate the situation be consistent and defined at the correct level of detail. This is accomplished using both the Scenarization Vocabulary (introduced in the current section) and the Specification Schemas (presented in the next section). As for the comprehension model, a set of graphs and the Scenarization Vocabulary constitute the Scenarization Model. The main scenarization concept types are:

- Agent: a Simulation Object Entity (or Actor) who voluntarily, under some Motivations Rules, performs Actions trying to modify the situation and, whose Reactions, under some "Reflex Rules," modify the situation.

- Patient: a Simulation Object Entity (or Actor) who never acts under Motivations Rules but whose Reactions, under some "Reflex Rules," modify the situation.
- Decor: a Simulation Object Entity part of the environment (not Actor) that is used by the Actor but never modified.
- Action: a Simulation Object Behaviour triggered by an Agent.
- Reaction: a Simulation Object Behaviour triggered by an Actor (Agent or Patient).
- Variable: a Simulation Object Attribute changed by the simulation execution, also called a dynamic property.
- Parameter: a Simulation Object Attribute that is not changed by the simulation execution but can be changed by the designer, also called a static property.
- Predicate: a Simulation Object Proposition constructed with a Simulation Object Entity, a relation and either an Object Attribute or a Simulation Object Behaviour.

Although closely related to these concepts, the main relation types will be presented in the next section, since they are not relevant to the current activity of identifying simulation objects.

Although they serve different purposes, the Comprehension Model and the Scenarization Model are closely related. Any meaningful simulation object part of the Scenarization Model implements a comprehension concept. Some comprehension elements will not be part of the Scenarization Model, either because they are not relevant to the simulation or they are not mature enough to be integrated at the current stage of the spiral process. Conversely, some elements will be introduced in the Scenarization Model to add details required for the actual execution of the simulation. In other words, the designer needs to isolate comprehension concepts and relations that are relevant for simulation and add concepts and relations required for simulation purposes. Figure 2 shows comprehension graph concepts linked under the generic Scenarization Vocabulary. For each comprehension graph element, the designer must determine whether or not this element will be part of the simulation and how it will fit into the generic Scenarization Vocabulary. Figure 3 shows a vocabulary resulting from this activity. The generic Scenarization Vocabulary builds on previous work on planning systems e.g., STRIPS in Nilsson (1980) and SAIRVO in Lizotte (1989).

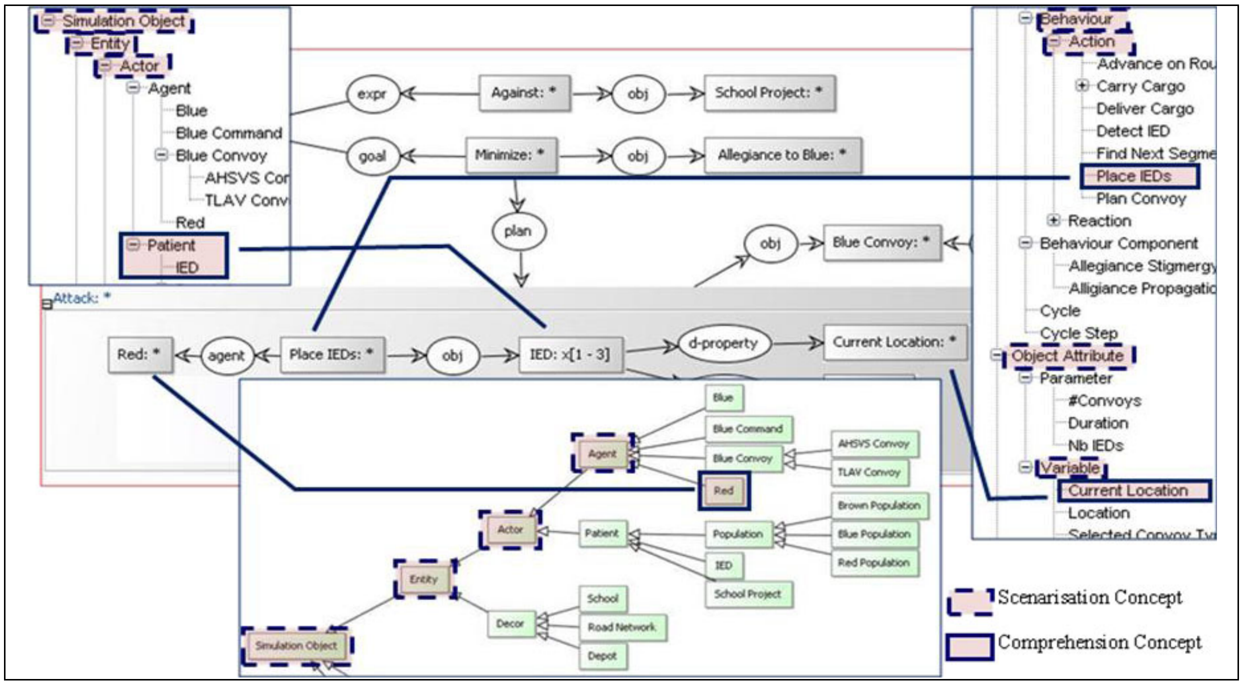

Figure 2: Linking comprehension concepts with the generic Scenarization Vocabulary

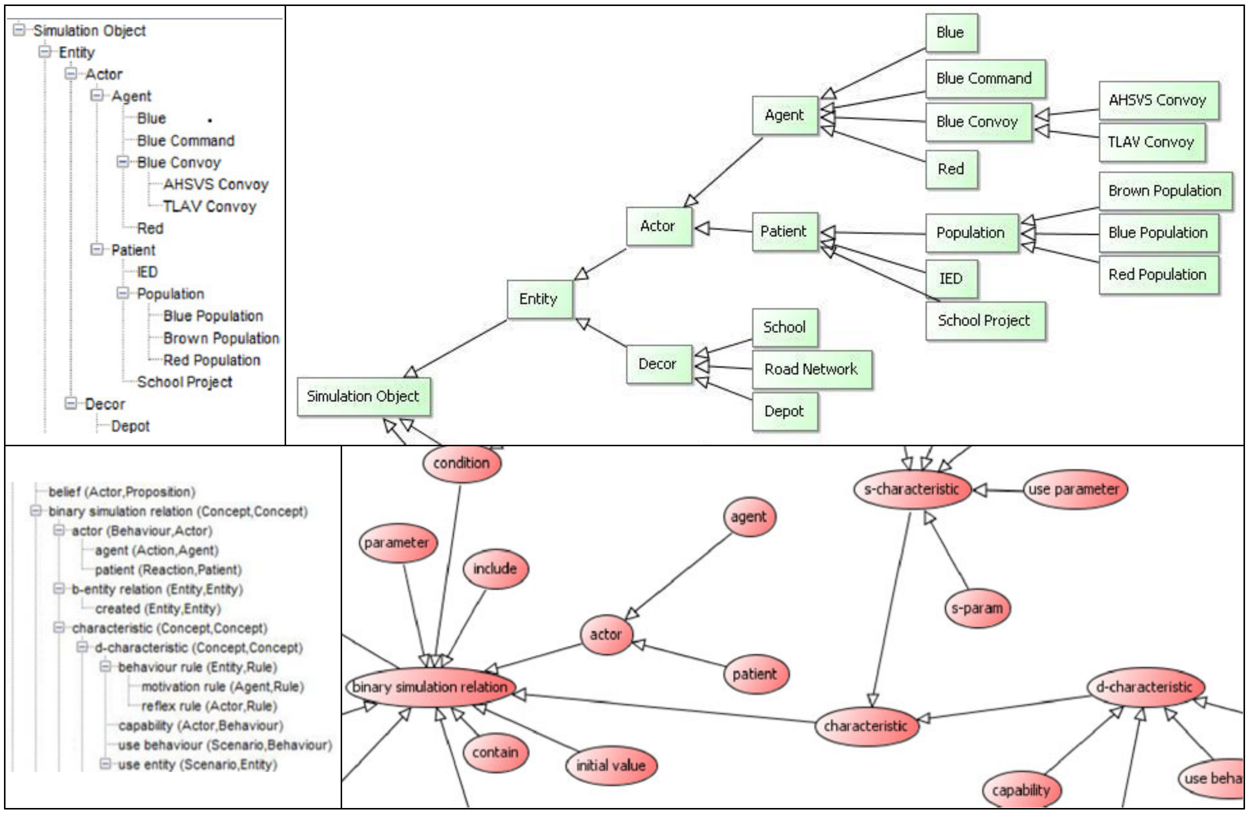

Figure 3: Specialization of the generic Scenarization Vocabulary

4 SCHEMAS: WORKING OUT THE SCENARIO MISE-EN-SCÈNE

After a key simulation concept (such as an agent or an action) is identified, the designer can start describing it using a specification schema. Specification schemas (also called prototypics) are associated with the following scenarization concept types: Agent, Patient, Decor, Behaviour and Scenario. The schema notion is defined in Chein and Mugnier (2008). In addition to specification schemas, construct schemas (also called patterns) facilitate the designer's work. Construct schemas are not about adding relations with other concepts to a specific concept type, but rather about a nested graph pattern that can be used to detail a concept. For instance, a precondition to a behaviour can be detailed using the attribute predicate schema "Simulation Object:*→attribute→Object Attribute."

Figure 4 shows an example of such a graph The "root" graph linking simulation elements follows the Scenario schema. It identifies agents, patients and a decor that are part of the scenario E, and specifies the different cycle step behaviours of the main cycle. A cycle can also include subcycles through its cycle steps. Cycle steps point to a subcycle through the "repeat" relation the same way the scenario concept points to the main cycle.

Within a cycle, the steps are executed according to their markers, e.g. main [1], main [2], …, main[n]. In addition to this static scheduling mode requiring a predefined sequence, a dynamic scheduling mode using stimulation (motivation and reflex) rules is available. The stimulation rules described below are associated with agents and patients. Reflex rules are triggered by a dynamic property change or actor addition/removal, whereas motivation rules are triggered according to their priority when the associated condition is satisfied.

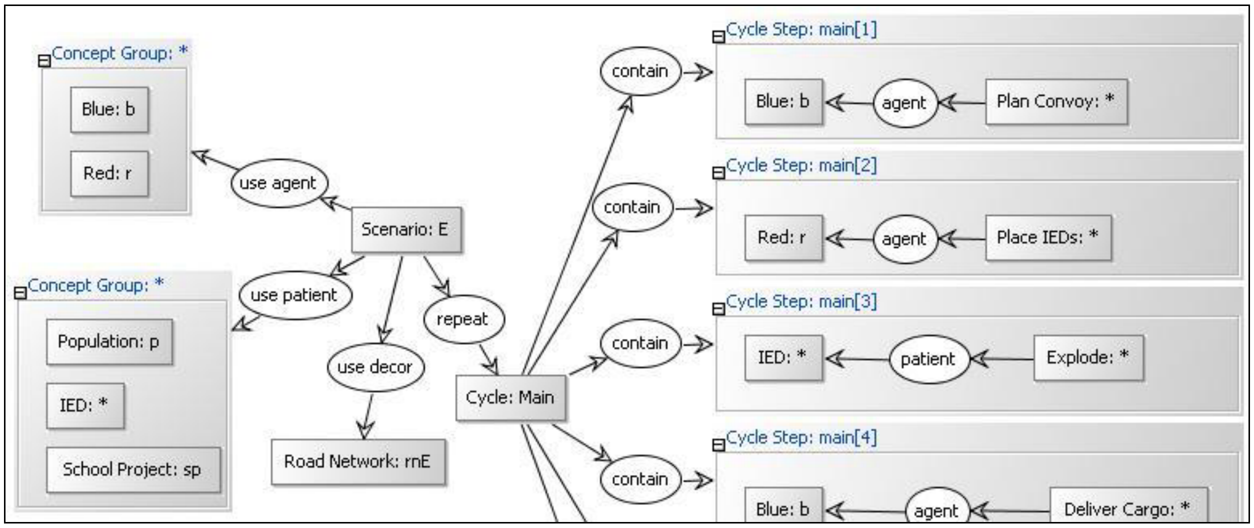

Figure 4: Specification of a Scenario

Figure 5 presents how to use an agent schema along with the resulting graph specifying the agent IED. In this schema and the following ones, many relations are utilized. Following are the main relations utilized by the entity schemas (Agent, Patient and Decor):

- s-property: associates a Parameter to a Simulation Object
- d-property: associates a Variable to a Simulation Object
- capability: associates a Behaviour to an Actor
- reflex rule: associates a triggering condition, to be verified if the situation changes, to a Reaction
- motivation rule: associates a triggering condition, to be verified at each simulation cycle, to an Action

Three additional relations meant to associate current beliefs, goals and plans to an Agent were imagined, but not experimented until now.

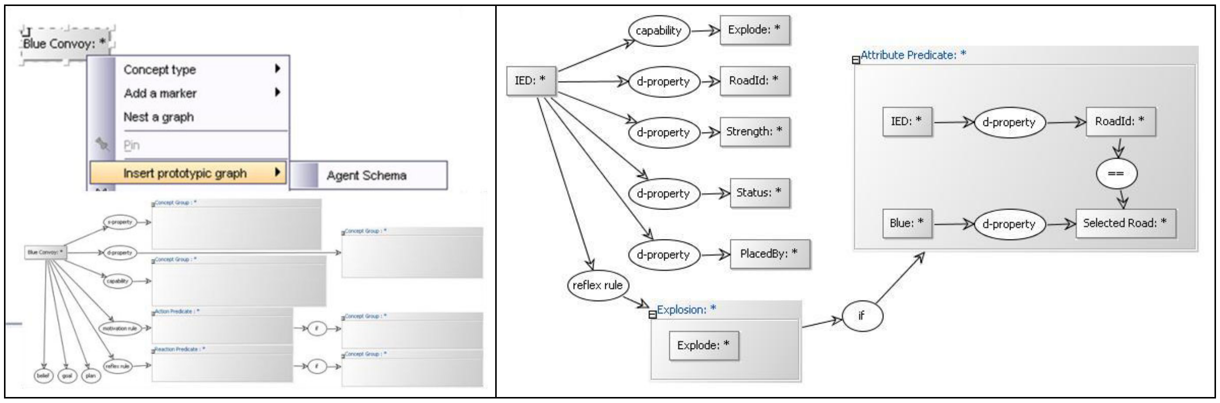

Figure 5: Specification of an Agent

A Behaviour Schema works in the same way, but uses different relations:

- precondition: associates a Proposition (e.g. Predicate), that must be true, with a Behaviour for the Behaviour to be feasible and actually begin.
- b-effect: associates a situation change to a Behaviour when the Behaviour begins.
- co-condition: associates a Proposition to a Behaviour, that must be true, during execution of the Behaviour. It must be verified at each simulation cycle between the beginning and the end of the Behaviour.

- e-effect: associates a situation change to a Behaviour when the Behaviour ends.

A sub-behaviour relation was imagined but not experimented until now. The idea was to decompose the Behaviour into a "time" constrained graph using "before" and "while" relations. However, this need is partially addressed using a stimulation rule including a set of behaviours ordered with the "before" relation.

5 JAVA SIMULATION FRAMEWORK: COMPLETING THE IMPLEMENTATION OF THE SIMULATION DYNAMICS

The IMAGE-SCE approach would not be complete without its agent-based simulator framework. Use of this Java framework is the last activity a designer must complete before actually running a simulation. From a root graph scenario (see previous section), the designer triggers the creation of the basic simulator software package. This transformation process uses the conceptual graphs (reachable from the root graph scenario) as input and produces implementation source code files specific to the problem. Added to the other files part of the framework, they constitute the basic simulator software package. The simulation engine of the framework is an adaptation of an interactive simulation software known as Multichronia (Rioux, Bernier, and Laurendeau 2008).

Most of the generated code is complete and does not need any additional work by the designer, though some does. The designer will find in this package a set of implementation classes, including method stubs, used mainly to detail relevant simulation dynamics. These are behaviour, stimulation and simulation cycle classes. Although there are plans to improve graph notation and minimize code writing by the designer, this stub approach was chosen deliberately for two main reasons. It allows a maximum of flexibility enabling any code insertion of interest to the designer. In addition, the graph notation exists to ease the designer's work and, in many cases, a behaviour is more easily expressed with a programming language rather than a graph notation.

5.1 Simulation Cycle Classes

An implementation class is generated for each Simulation Cycle declared in the Scenario graph. A cycle includes a set of iterations. Each iteration either triggers a set of cycle steps (in a static scheduling mode) or a set of stimulation rules (in a dynamic scheduling mode). As presented in Table 1, four method stubs are available to the designer.

Table 1: Cycle Method Stubs

Method Name	**Description**
beforeCycle	Can be used to perform initialisation before a Cycle starts
cycleFinished	Used to implement a condition to stop the cycle
beforeIteration	Can be used to execute code before an iteration
afterIteration	Can be used to execute code after an iteration

5.2 Entity (Actor, Patient and Decor) Classes

An implementation class is generated for every simulation entity type declared in the scenario graph. Table 2 lists the available methods.

5.3 Stimulation (Motivation and Reflex) Classes

An implementation class is generated for every stimulation declared in the scenario graph. Table 3 shows the available method stub.

Table 2: Entity Methods

Method Name	Description
new<EntityType>()	A single static method creating an Actor
get<ParameterName>()	Getter methods for each Actor's parameter
get<VariableName>() *set<VarableName>()*	Getter and Setter methods for each Actor's variable
get<BehaviourName>()	A getter method for each Actor's behaviour
get<StimulationName>()	A getter method for each Actor's stimulation
getID()	A getter method for the Actor's ID
get(<Entity ID>)	Static method returning an entity identified by <Entity ID>
GetList()	Static method returning the collection of created Actors

Table 3: Stimulation Method Stub

Method Name	Description
stimulationIsTrue	Current situation stimulates the Actor to perform the associated behaviour

This stimulation system allows Actors minimal autonomy, but further work in the framework is required in order to remove the need to declare stimulations in the scenario graph. However, the implementation already generates adequate code for stimulation (motivation and reflex) rules using basic comparison operators (if, if not, ==, !=, <, <=, >, >=) and the designer may always overwrite the method.

5.4 Behavioural Classes

An implementation class is generated for every simulation entity type declared in the scenario graph. Table 2 lists available method stubs for use by the designer. The framework allows a behaviour to begin and end in a single simulation time step or span many steps.

Table 4: Behaviour Method Stubs

	Method Name	Description	Execution
	Init	Performs initialisations e.g. setting parameters	On the Behaviour creation
Single and multi-steps	*preconditionsAreTrue*	Current situation allows to begin execution	Situation satisfies a motivation or a reflex condition
	beginExecution	Performs effects resulting from the execution start	If preconditions are satisfied
	applyEffects	Performs effects resulting from a successful execution	If execution is finished
Multi step only	*executionFinished*	Current situation required to successfully stop the execution	On a new cycle
	coconditionsAreTrue	Current situation allows to pursue execution	If execution is not finished
	continueExecution	Performs recurrent effects for the current cycle	If coconditions are satisfied
	abortExecution	Performs effects resulting from an interrupted execution (e.g. undo effects performed at the execution start)	If coconditions are not satisfied

6 THE SPIRAL PROCESS: RUNNING AND REVISITING THE SIMULATION

The previous sections presented the different activities a designer must accomplish in order to build a functional agent-based simulator. This last section, describing the methodology, presents how these activities are used to actually reach a validated simulation. Table 5 is an attempt to show how the IMAGE-SCE approach supports tasks and steps identified by Macal and North (2006):

Tasks
- (a) Identification of agents and a behaviour theory
- (b) Identification of agent relationships and an interaction theory
- (c) Collection of agent-related data
- (d) Validation of the models

Steps
- (1) Agent Identification (agent types and other objects along with their attributes)
- (2) Environment Definition
- (3) Agent Methods Specification (agent attributes updates in response to interactions)
- (4) Agent Interactions Control (which agents interact, when, and how)
- (5) Software Implementation

Table 5: IMAGE-SCE vs. Macal and North (2006) tasks and steps

IMAGE-SCE Component	**IMAGE-SCE Tool**	**Task**	**Steps**				
			(1)	(2)	(3)	(4)	(5)
Representing the problem	Graph Notation	(a) (b)					
Identifying simulation objects	Vocabulary	(a)	Agent, Patient, Object Attribute …	Decor, Object Attribute …			
Working out the mise-en-scène	Schemas				Agent, Patient, Behaviour schemas and Java code …	Scenario, Cycle Step, Motivation Rule, Reflex Rule …	
Implementing the dynamic	Java Framework						Behaviour Java code, BeforeCycle, BeforeIteration …
Running & Revisiting the simulation	Spiral Process	(d)					

Figure 6 provides an example of an insurgency simulator evolution during a spiral process.

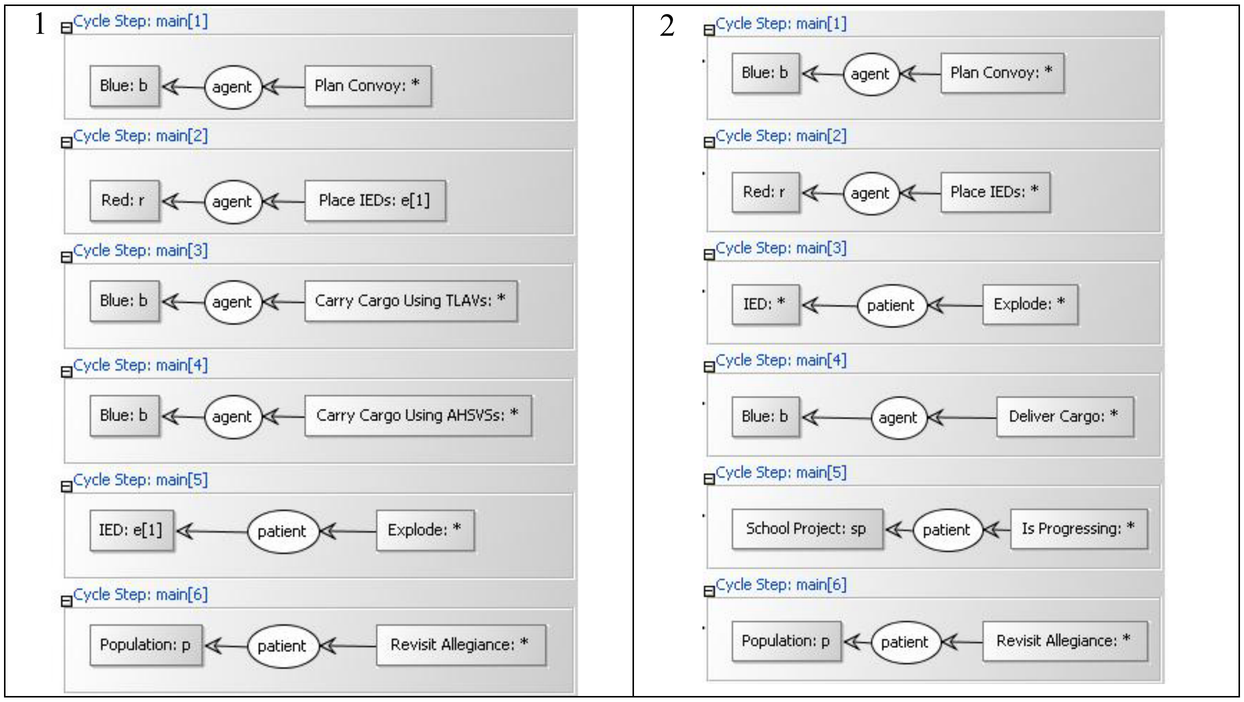

Figure 6(a): Increment 1 and 2 of the insurgency scenario

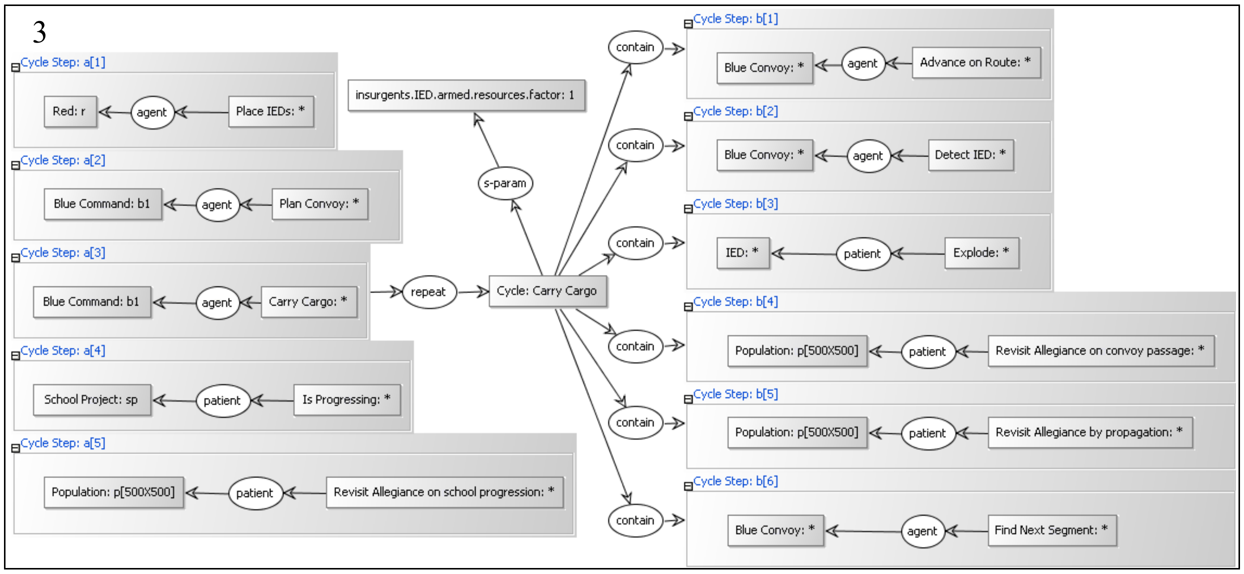

Figure 6(b): Increment 3 of the same insurgency scenario

```
…
------------- 4 - END OF ITERATION ------------
END OF CYCLE ?
-< Main > - < cycleFinished >-
        Cumulated Cargo < Required Cargo
---------- 5 - BEGINING OF ITERATION ----------
-< Main > - < beforeIteration >-
        Blue Carried Cargo          = 0.0
        Blue Selected Convoy Type    =
        Blue Selected Road          =
        Red Resource Gain           = 0.0
        School Project New Cargo      = 0.0
ACTOR 'Blue(b)' DOING 'PlanConvoy'
-< PlanConvoy > - < preconditionsAreTrue >-{YES}
-< PlanConvoy > - < beginExecution >-
-< PlanConvoy > - < coconditionsAreTrue >-
-< PlanConvoy > - < endExecution >-
-< PlanConvoy > - < applyEffects >-
        Selected Convoy Type = TLAV Convoy
        Carried Cargo      = 2.0
        Selected Road      = 3
ACTOR 'Red(r)' DOING 'PlaceIEDs'
-< PlaceIEDs > - < preconditionsAreTrue >- {YES}
-< PlaceIEDs > - < beginExecution >-
-< PlaceIEDs > - < coconditionsAreTrue >-
-< PlaceIEDs > - < endExecution >-
-< PlaceIEDs > - < applyEffects >-
        RoadID = 1
        Strength = 0.11374688
ACTOR 'IED(The IED)' DOING 'Explode'
-< Explode > - < preconditionsAreTrue >- {NO}
ACTOR 'Blue(b)' DOING 'DeliverCargo'
-< DeliverCargo > - < preconditionsAreTrue >- {YES}
-< DeliverCargo > - < beginExecution >-
-< DeliverCargo > - < coconditionsAreTrue >-
-< DeliverCargo > - < endExecution >-
-< DeliverCargo > - < applyEffects >-
        School Project New Cargo      = 2.0
ACTOR 'School Project(sp)' DOING 'IsProgressing'
-< IsProgressing > - < coconditionsAreTrue >- {YES}
-< IsProgressing > - < continueExecution >-
        School Project Cumulated Cargo = 11.0
ACTOR 'Population(p)' DOING 'RevisitAllegiance'
-< RevisitAllegiance > - < preconditionsAreTrue >-{YES}
-< RevisitAllegiance > - < beginExecution >-
        Population Allegiance To Blue = 9.0
-< RevisitAllegiance > - < coconditionsAreTrue >-
-< RevisitAllegiance > - < endExecution >-
-< RevisitAllegiance > - < applyEffects >-
-< Main > - < afterIteration >-
------------- 5 - END OF ITERATION ------------
END OF CYCLE ?
-< Main > - < cycleFinished >-
        Cumulated Cargo >= Required Cargo
-< Main > - < afterCycle >-
```

Figure 7: A simulator execution trace

7 CONCLUSION

The paper introduces a computer-assisted approach to develop an agent-based simulator from the initial problem definition. This approach includes five (5) components:

(1) A graphical notation, extending the conceptual graph formalism, to conceptually represent the problem of interest without any predefined ontology, i.e., using any concepts and relations deemed necessary by the designer.

(2) A predefined generic scenarization vocabulary describing the various roles that a concept or relation can play in the agent-based simulation.
(3) A predefined set of specification schemas enabling the description and linking of simulation elements.
(4) An agent-based simulator Java framework allowing the designer to complete the implementation of the simulation dynamics.
(5) A spiral process using the elements above to iteratively and incrementally build and validate the simulator.

For many problems, such an approach will most likely decrease the effort required to build an agent-based simulator. Moreover, integrated in the overall IMAGE concept, it enables a team of experts to collaboratively work out a valid simulator. The IMAGE context (Lizotte et al. 2008) and Scenarization approach is summarized in Figure 8. The IMAGE concept, which includes a Scenarization part, targets the collaboration of experts trying to reach a shared understanding of a complex situation. The IMAGE-Scenarization approach (IMAGE-SCE) uses Comprehension graphs and a generic Scenarization Vocabulary to produce a Scenarization Model and a Java Simulator.

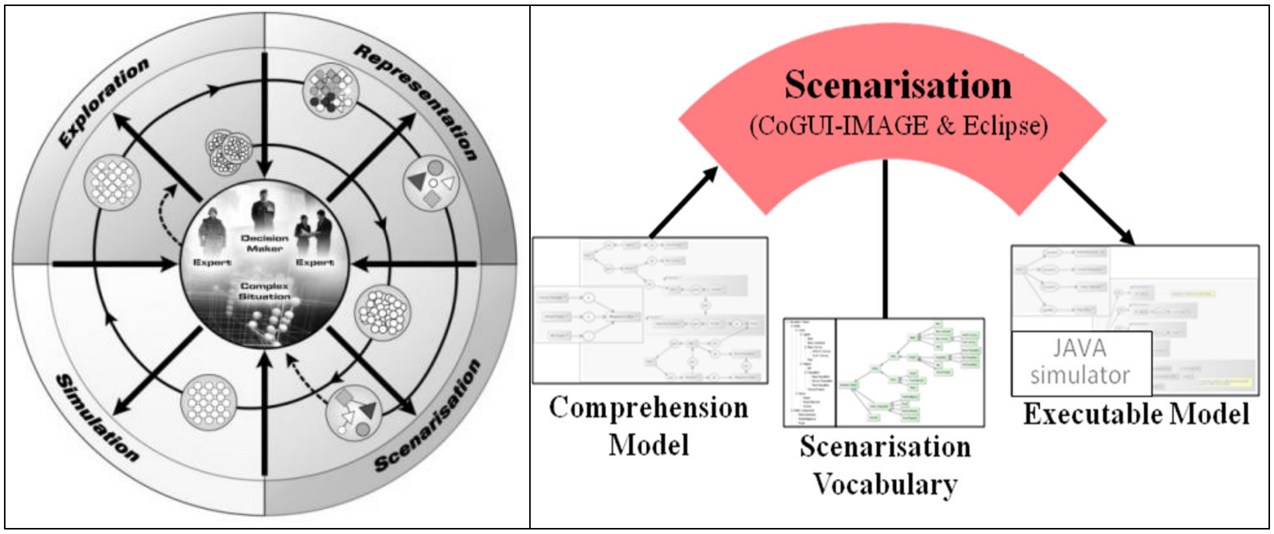

Figure 8: IMAGE-Scenarization approach

Like the whole IMAGE concept, this Scenarization approach will likely be refined in coming years through prototyping and fine-tuning to specific use priorities. Enhanced usability and automation through graph notation (~visual programming) will certainly be considered, along with better support of proposition nested graphs.

REFERENCES

Chein, M., and M.L. Mugnier. 2008. *Graph-based knowledge representation: computational foundations of conceptual graphs*. Springer.

Genest, D. 2010. *Cogitant Reference Manual version 5.2.3*, Available via <https://cogitant.source.forge.net/files/cogitant.pdf> [accessed April 12, 2010].

Lizotte, M., and B. Moulin. 1989. SAIRVO: A planning System which implement the Actem concept. *Knowledge-Based Systems Journal* 2(4):210-218.

Lizotte, M., D. Poussart, F. Bernier, M. Mokhtari, E. Boivin, and M. DuCharme. 2008. IMAGE: simulation for understanding complex situations and increasing future force agility models. In *Proceedings of the Army Science Conference 2008*, Orlando, Florida.

Macal, C., and M. North. 2006. Tutorial on agent-based modeling and simulation part 2: how to model with agents. In *Proceedings of the 2006 Winter Simulation Conference*, eds. L. R. Perrone, F. P. Wieland, J. Liu, B. G. Lawson, D. M. Nicol, and R. M. Fujimoto, 73-83. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Nilsson, N.J. 1980. *Principles of artificial intelligence*. Palo Alto, California: Tioga Pub. Co.

Rioux, F., F. Bernier, and D. Laurendeau. 2008. Multichronia – a generic parameter, simulation, data, and visual space exploration framework, *Interservice/Industry Training, Simulation & Education Conference*, Orlando, Florida.

Sowa, J., 1984. *Conceptual structures: information processing in mind and machine*, Addison Wesley, Reading Mass.

Van Harmelen, F., V. Lifschitz, and B. Porter. 2008. *Handbook of knowledge representation*, Elsevier.

AUTHOR BIOGRAPHIES

MICHEL LIZOTTE has been working at Defence Research and Development (DRDC) – Valcartier since 1999. He is currently the Group Leader of the Simulation and Comprehension of Complex Situations group. Previously, he was an information technology (IT) consultant for almost 11 years including eight at DMR. During this period, the vast majority of his assignments were carried out for research establishments such as the National Research Council (NRC). He obtained his Bachelor's Degree in Computer Science (1984) and his Master's Degree in Artificial Intelligence (1988) from Université Laval (Québec). His current interests and recent work encompass approaches to understanding complex situations and software-intensive system architecture, software and military capability engineering, and software and system architecture. His email address is <Michel.Lizotte@drdc-rddc.gc.ca>.

FRANÇOIS RIOUX received a B.Eng. Degree in Electrical Engineering from Laval University in 2003. He received a M.Eng. Degree from McGill University in 2005. He received his Ph.D. in Electrical Engineering from Laval University in 2009. He now works as a consultant for the LTI Software and Engineering firm. He performs research in collaboration with Defence R&D Canada – Valcartier on the topics of interactive simulation and visualization applied at better understanding complex systems. His email address is <frioux@ltinfo.ca>.