Situations in Simulations: An Initial Appraisal

Alessandro M. Baldi, Patrícia Dockhorn Costa, Eduardo Zambon, João Paulo A. Almeida
Computer Science Departament
Federal University of Espírito Santo (UFES)
Vitória, Brazil
alessandromurtabaldi@gmail.com, pdcosta@inf.ufes.br, zambon@inf.ufes.br, jpalmeida@ieee.org

***Abstract*—Approaches to simulation in the literature range from conventional object-orientation programming, which dates back to the seminal Simula programming language, to the more recent graphical modeling environments in the scope of agent-based modeling and simulation. Despite their clear impact and benefits, we have observed that these approaches have not yet provided support for situation lifecycle management, and do not address a notion of "situation" explicitly. In this paper, we perform a case study in order to examine how the various approaches support the design and execution of simulations. We are particularly interested in contrasting existing approaches to simulation with a declarative rule-based approach that supports situations explicitly. We take into account design time concerns as well as execution time performance. The study focuses on the simulation of populations of Aedes aegypti mosquitoes in urban environments.**

***Index Terms*—simulation approaches, situations, comparison**

I. Introduction

Simulation-based approaches have been employed successfully for the purpose of study in a number of domains [1]–[3]. These approaches share the aim to investigate the appropriateness of responses to various situations in a field of action, constituting an important tool for smart decision support. Since they can mimic real-world situations, simulations allow for experimentation under a range of conditions, enabling (i) the identification of possible issues in advance; (ii) the training of systems and humans; (iii) the analysis of possible system states and outcomes, etc.

Despite their similar goals, the various simulation approaches have utilized an array of styles for the specification of expected simulation behavior as well as various means for simulation execution. For example, some approaches to simulation in the literature employ conventional object-oriented programming, a practice which can be traced to the seminal Simula programming language. Other approaches propose frameworks or libraries dedicated to simulation [4], enhancing object-oriented approaches with reuse of basic simulation functionality, thereby facilitating the development of simulation programs. More recently, graphical modeling environments in the scope of agent-based modeling and simulation [5] have been proposed, providing reuse of basic mechanisms for simulation, such as support for spatial and temporal dimensions and their visualization, and raising the level of abstraction of simulation specification and execution.

We have observed that, despite the importance of the notion of "situation" in the comprehension of the real-world [6], the existing range of approaches to simulation have not yet addressed a notion of "situation" explicitly. This has motivated us to investigate whether support for what we call situation lifecycle management [7] could have a beneficial role in simulations. This paper reports on the current state of this investigation, with an initial appraisal of the role of situations in simulations.

We explore four different approaches to simulation and present a quantitative evaluation concerning run-time performance. Our aim is to examine their suitability for simulation considering their ease of use and feasibility as a computational tool. Based on the evaluation, we also aim to issue recommendations for further development of situation-aware simulation environments.

We have realized the same simulation scenario in four ways: a conventional object-oriented implementation in Java, a model-based graph-transformation specification in GROOVE [8], an agent-based modeling and simulation specification in RePast [5], and a declarative rule-based situation-aware implementation in Drools/SCENE [7].

The case study involves the simulation of a large number of mosquitoes throughout their lives, and thus we pay particular attention to the time performance of the simulation. Since we envision in the medium term the use of a graphical model-based frontend and a (declarative) rule-based backend to build a comprehensive simulation environment, we are particularly interested in answering whether graph-based and rule-based approaches are able to scale in scenarios involving many situation elements (in our case, the individual mosquitoes as vectors of diseases.) We are also particularly interested in contrasting existing approaches to simulation with a declarative rule-based approach that supports situations explicitly.

The paper is further structured as follows: Section II presents the Aedes aegypti simulation scenario, Section III describes the four realizations of the scenario, Section IV contrasts the various approaches, Section V discusses related work and Section VI presents our main conclusions and outlines some future work.

II. The Aedes aegypti Simulation Scenario

The Aedes aegypti is a mosquito present mainly in the urban environment of tropical and subtropical countries and is the main vector of Zika, Dengue, Chikungunya and Yellow Fever diseases [9]. The dynamics of Aedes aegypti is complex and has motivated researchers to conduct several studies in various areas [10], [11].

The emergence and resurgence of diseases transmitted by Aedes aegypti remain a major threat [12], [13]. The expectation of researchers in the area is that reliable and useful predictive measures can be developed with the help of efficient and appropriate tools [10], [13]. With this mindset, we have defined in this paper an Aedes aegypti simulation scenario that mimics the dynamics of the mosquito in a urban environment. We believe that the result of this simulation is an important decision support tool since it can assist health professionals to identify and predict places with higher incidences of Aedes aegypti [14].

We have considered the most common and important elements of the Aedes aegypti behavior in a urban environment, which are: reproduction of mosquitoes in foci, mosquito population growth and vector combat through traps. The simulation scenario consists of houses, mosquitoes, eggs, vector control agents and traps. Houses represent the places where mosquitoes can oviposit (lay eggs) and may have foci (mosquito breeding sites) or traps. In case a female mosquito oviposits in foci, the eggs evolve to new mosquitoes after 20 days [15]. Otherwise, trapped oviposition kills mosquitoes and the eggs do not evolve into new mosquitoes [13], [16]. A health professional checks traps two days after oviposition and observes if there are eggs and if a mosquito vector control is necessary in the region where the trap is located [17]. The control consists of smoke poison and home visit, which causes extermination of eggs, mosquitoes and foci [17]. The surviving mosquitoes remain doing the oviposition for up to 37 days [15]. During their lifetime, mosquitoes are free to fly and lay eggs up to 100 meters from their point of birth [17]. Climatic phenomena are also considered in the scenario by means of rain simulation, bringing back foci.

The spatial aspects are taken from the campus of the Federal University of Espírito Santo, in Brazil. The campus covers 157 ha of area, of which 104 ha is constructed area. We have used OpenStreetMaps to gather geolocation information of all buildings on campus [18]. We have used public health data that has been published about the cities of Vitória and Rio de Janeiro. In order to establish the trap density, we employed a study carried out in Vitória which found 1410 traps distributed among 80 neighborhoods in this city in 2011 [19] (an average of about 17 traps per neighborhood). Considering the university campus as a single neighborhood, we have established for the simulation scenario the use of 17 traps throughout the campus. As for the mosquitoes population density, we have used data from a study carried out in Rio de Janeiro [20], in which a population density of 10 female Aedes aegypti per hectare was estimated. Since the university campus has 104 ha of constructed area, we have considered 1040 mosquitoes as the initial mosquito population.

III. Simulation Implementation

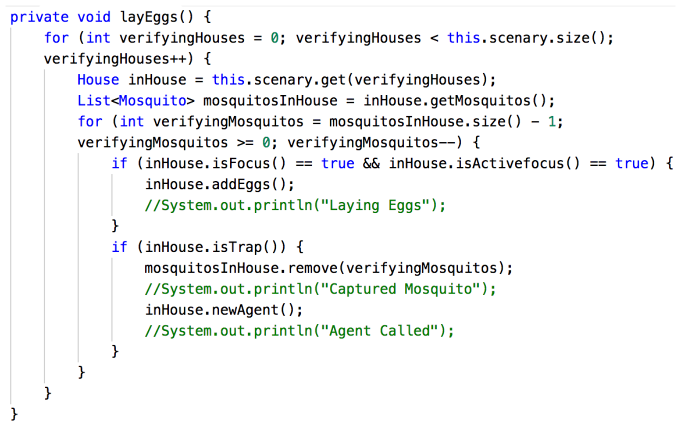
We have implemented our scenario in four simulation approaches, as described in the next subsections.

A. *Object-oriented implementation in Java*

A first implementation was performed in Java, as a representative of a high-level, object-oriented, and general-purpose language. The implementation of the simulation in Java is given through the manipulation of objects. Objects are formed by attributes describing their properties according to the domain in the real world. The attributes are then manipulated in if-then-else nested conditionals in order to implement object behaviors.

The implementation scenario defines a list of objects of type “House”, which are connected to other house objects, representing the direct links between houses in which mosquitoes can migrate. An object of type “House” may also contain a reference to lists of objects of type “Mosquito”, “Eggs” and “Agent”, indicating whether they have mosquitoes, eggs or agents in that house, respectively. A number of attributes are used to represent the state of each object (e.g., whether mosquitoes have already flown or oviposited in the current simulation day).

The Java simulation progresses in rounds by means of loops, in which the simulation behaviors are executed as if-then-else statements. The code snipped in figure 1, depicts a particular behavior implemented by the Java simulation, in which mosquitoes lay eggs. This code checks whether there are houses with active foci (by means of the “activefocus” boolean attribute) and, if this is the case, the method “addEggs()” is invoked to insert eggs in those particular houses.



```java
private void layEggs() {
    for (int verifyingHouses = 0; verifyingHouses < this.scenary.size();
    verifyingHouses++) {
        House inHouse = this.scenary.get(verifyingHouses);
        List<Mosquito> mosquitosInHouse = inHouse.getMosquitos();
        for (int verifyingMosquitos = mosquitosInHouse.size() - 1;
        verifyingMosquitos >= 0; verifyingMosquitos--) {
            if (inHouse.isFocus() == true && inHouse.isActivefocus() == true) {
                inHouse.addEggs();
                //System.out.println("Laying Eggs");
            }
            if (inHouse.isTrap()) {
                mosquitosInHouse.remove(verifyingMosquitos);
                //System.out.println("Captured Mosquito");
                inHouse.newAgent();
                //System.out.println("Agent Called");
            }
        }
    }
}
```

Fig. 1. LayEggs Code

The complete implementation is omitted here due to space constraints and can be found in [21].

B. Graph-based implementation in GROOVE

A second implementation employed the GROOVE (GRaph based Object-Oriented VErification) tool-set. GROOVE uses graphs to represent the states of the system and applies graph transformation (rewrite) rules to model system dynamics [8]. The tool has an editor for creating state graphs and rewrite rules, a simulator to visualize the transformations in the system state and a generator to explore state spaces automatically [22].

The graphs manipulated by GROOVE are formed by nodes and directed binary edges. Edges are annotated with *labels* that characterize the relationship between the two connected nodes. Additionally, we can have attributes (integer and Boolean values) that describe certain properties of each node. To model a simulation in GROOVE, we need to implement the initial scenario as a *start state graph*.

State graphs are transformed in GROOVE by graph transformation rules. A rule is also a graph, but, in contrast to state graphs, rule graphs are subdivided into subgraphs LHS and RHS. Subgraph LHS can be seen as the precondition for the rule application, and subgraph RHS as the effect of the application. A rule is applied on a host graph (that is, the current state graph that will undergo the transformation). This application requires finding a match of LHS in the host graph, and then replacing this match with a copy of the rule RHS, thus forming a new state graph [8].

Figure 2 shows the syntax of rule graph elements in GROOVE. The role of nodes and edges in a rule is identified by the color and dash type on the edge and around the node. Fine-dashed (blue) elements (the edge labeled “edge2” in Figure 2) are elements that must be present in the current state of the simulation and that are removed when the rule is applied. Similarly, elements with fine continuous (grey) lines (“edge1” and all the nodes) are also elements that must be present in the current state, but these are preserved by the rule application. Thick-dashed (red) elements (“edge4”) cannot occur in the current state, otherwise the rule is not applicable. Finally, elements in a thick continuous (green) line (“edge3”) are created when the rule is applied.

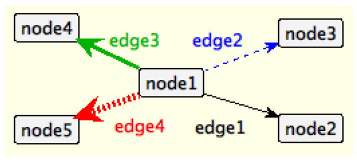


Fig. 2. Rule graph syntax in GROOVE

The scenario elements were modeled in GROOVE using nodes with proper labels. The **house** nodes (with their structure shown in figure 3-1) has **fly**-labeled edges linking other **house** nodes, indicating that a mosquito can fly from one house to another. There is also the possibility of a house containing **eggs** (figure 3-2) or a **mosquito** (figure 3-3). A **mosquito** node has an edge indicating the house where the mosquito was born, and two Boolean attributes indicating whether the mosquito has already performed oviposition and whether it has flown in the current round of the simulation. In addition, each **mosquito** has an integer attribute indicating its age in days. The **agent** nodes in figure 3-4 must always be in some house and also have a visit counter for each visited house. The simulation scenario is done in rounds, during which all the actions of the mosquitoes and agents are carried out. At the end of each round, a day counter is incremented until a given limit is reached.

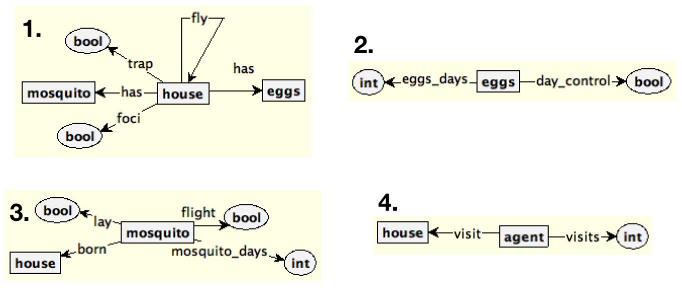


Fig. 3. Scenario elements modeled in GROOVE

Figure 4 represents the rule that captures the condition for oviposition and the corresponding action. If a house has foci and a mosquito that has not yet layed eggs (“lay” is false), a batch of eggs is created to represent oviposition. This is done with the creation of an “eggs” node connected to the house and initialized with a day count of 1. The complete set of rules is omitted here due to space constraints and can be found in [21].

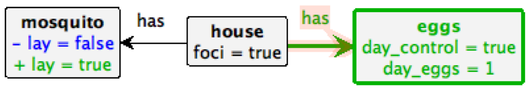


Fig. 4. An example rule: oviposition

C. Rule-based Implementation in SCENE

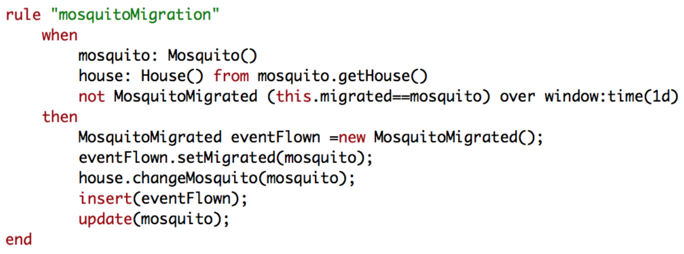
A third implementation was performed using SCENE [7]. SCENE is a platform that runs on the top of JBoss Drools [23] engine with the aim of supporting rule-based situation-awareness, enabling the specification of situations and maintaining the lifecycle of situations [7]. The Aedes aegypti SCENE simulation behavior is defined in terms of Drools rules and Situation rules.

A Drools rule declaration comprises a condition and a consequence expression block, respectively referred to as Left Hand Side (LHS) and Right Hand Side (RHS). A rule specifies that when the particular set of conditions defined in the LHS occurs, the list of actions in the RHS should be executed. The LHS is composed of conditional elements which can be combined through logical operators, such as **and**, **or**, **not** and **exists**; and set operators, such as **contains** and **member of**. A conditional element can be a pattern or a constraint. A pattern matches against a fact in the working memory (of the specified class type); constraints match against properties, and are defined as conditions within a pattern. The RHS allows the declaration of procedural code to be executed when the conditions defined in LHS are satisfied.

Situation rules are particular types of Drools rules, in which it is possible to objectify situations and reason about them as first-class citizens in the specifications. Conditional patterns defined in the LHS of the Situation Rule declaration establish the conditions which must hold for a situation of a particular situation type to exist. Situation lifecycle control (situation creation, activation and deactivation) is completely realized by SCENE.

We have also used the CEP (Complex Event Processing) functionality of Drools (Fusion) in order to specify the occurrences of events in the simulation. For example, we define the following as events: (i) episodes of rain are represented as instances of a Drools Fusion Event Type called "Rain", which is instantiated every 15 days; (ii) the migration of mosquitoes; (iii) oviposition and, finally, (iv) the incubation of eggs.

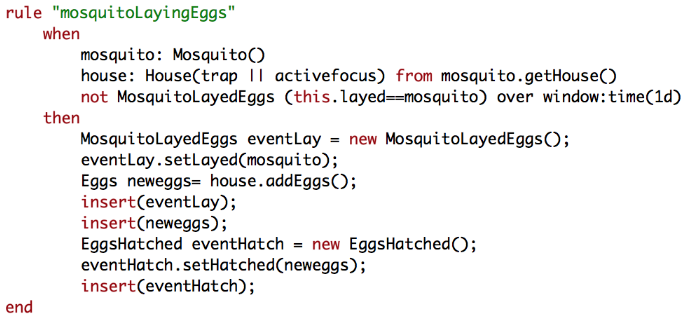
The simulation progress is defined by triggering of rules. For example, Figure 5 depicts the rule "mosquitoMigration", which defines the following behavior: if a mosquito is in a house but has not yet migrated in the past day (time constraint defined with the "over window:time" operator), generate a "MosquitoMigrated" event for that mosquito and change its location to a neighbor house (randomly chosen). This rule forces mosquito migration every single day, to a different house.



```
rule "mosquitoMigration"
    when
        mosquito: Mosquito()
        house: House() from mosquito.getHouse()
        not MosquitoMigrated (this.migrated==mosquito) over window:time(1d)
    then
        MosquitoMigrated eventFlown =new MosquitoMigrated();
        eventFlown.setMigrated(mosquito);
        house.changeMosquito(mosquito);
        insert(eventFlown);
        update(mosquito);
end
```

Fig. 5. Mosquito Migration Rule

Another example of rule that makes the simulation progress is the "mosquitoLayingEggs" rule (in Figure 6). The goal of this rule is to force oviposition of all mosquitoes located in a breeding site (houses with traps or active focus). Therefore, this rule is executed when there is no "MosquitoLayedEggs" event in the past day for a mosquito that is located in a house with traps or active focus. If this condition is met, a new event "MosquitoLayedEggs" is triggered for that mosquito and new eggs are inserted for that particular house.

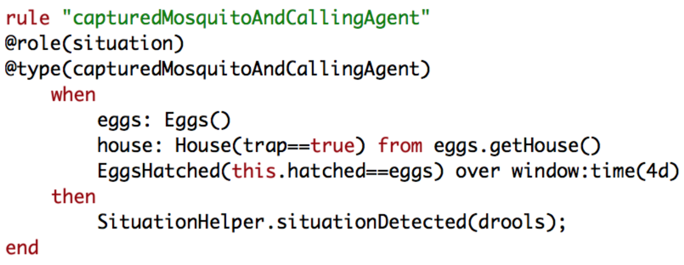


```
rule "mosquitoLayingEggs"
    when
        mosquito: Mosquito()
        house: House(trap || activefocus) from mosquito.getHouse()
        not MosquitoLayedEggs (this.layed==mosquito) over window:time(1d)
    then
        MosquitoLayedEggs eventLay = new MosquitoLayedEggs();
        eventLay.setLayed(mosquito);
        Eggs neweggs= house.addEggs();
        insert(eventLay);
        insert(neweggs);
        EggsHatched eventHatch = new EggsHatched();
        eventHatch.setHatched(neweggs);
        insert(eventHatch);
end
```

Fig. 6. Mosquito Laying Eggs Rule

The simulation of incubation of eggs is similarly specified. We define a rule "eggsHatching" in which the LHS detects whether there are eggs in houses with active foci that have not yet hatched for the past 20 days. The RHS of this rule simulates the birth of 10 mosquitoes in that particular house.

Figure 7 depicts the capturedMosquitoAndCallingAgent rule, which is an example of a Situation Rule in the simulation. It defines the situation in which mosquitoes have been detected in a trap but the extermination agents have not arrived yet. This situation is activated when an event of EggsHatched is detected for a particular house and remains active for over 4 days.



```
rule "capturedMosquitoAndCallingAgent"
@role(situation)
@type(capturedMosquitoAndCallingAgent)
    when
        eggs: Eggs()
        house: House(trap==true) from eggs.getHouse()
        EggsHatched(this.hatched==eggs) over window:time(4d)
    then
        SituationHelper.situationDetected(drools);
end
```

Fig. 7. Captured Mosquito and Calling Agent Situation

Both activation and deactivation events related to the situation capturedMosquitoAndCallingAgent are used by other rules. For example, the "AgentWorking" rule is executed when the deactivation of the capturedMosquitoAndCallingAgent situation is detected (i.e., 4 days after the eggs have hatched). The RHS of this rule simulates the work of an agent, which is the extermination of eggs and mosquitoes of that particular house and of 4 neighbor houses, randomly chosen.

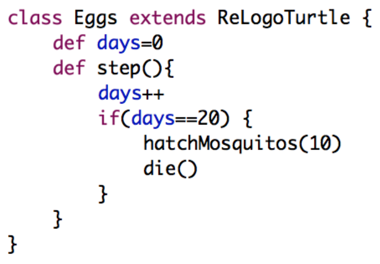
The complete code is available at [21].

D. *Agent-based modeling with RePast*

A final implementation was performed with RePast. RePast is a tool that uses the Agent-Based Modeling and Simulation (ABM&S) approach, in which agents exhibit independent and autonomous behaviors. A number of programming languages are supported by RePast, such as Logo (called reLogo), Java and C#. We have chosen reLogo for this simulation, which is the recommendation of various RePast papers since it is a simple and fast language to implement simulations.

In order to make simulations in RePast, developers should define agent types (or "individuals") and their behaviors. The running of the simulation is divided into time steps or "ticks", therefore, agent actions should be defined for each tick. RePast natively supports spatial functionality and also offers off-the-shelf agents behaviors. Simulations in RePast also require the configuration of the UserObservation component, which offers a graphical interface to define the initial configuration of the scenarios as well as defining shapes and colors to agents.

The following agent types have been defined in the RePast simulation: Agents, Eggs, House and Mosquito. Figure 8 depicts the definition of the "Eggs" agent type. In order to define this agent behavior, we should define their actions for each tick in the simulation, which is done by implementing the "step" method inherited from ReLogoTurtle class. For the Eggs agent type, in every tick of the simulation, the variable days is incremented by one and, when it reaches 20, the simulation creates 10 mosquito agents (representing the birth of 10 mosquitoes from that batch of eggs), and, in addition, this particular Eggs "agent" "dies". The complete code is available at [21].



```
class Eggs extends ReLogoTurtle {
    def days=0
    def step(){
        days++
        if(days==20) {
            hatchMosquitos(10)
            die()
        }
    }
}
```

Fig. 8. Agent "Eggs"

IV. Performance Results

Table I summarizes the results of the performance tests for each of the approaches. Simulation times corresponding to 5 to 90 scenery days are represented in each row. Results were obtained on a MacBook Pro (Retina, 13-inch, Early 2015) with Intel i5-5257U 2.7 GHz, 8GB 1867 MHz DDR3 memory, 128GB SSD and Mac OS High Sierra 10.13.3. The implementation in Java was found to have the best time performance, followed by Scene, RePast and GROOVE respectively. The dash marks ("–") represent the absence of measurements due to memory scalability issues as discussed in the sequel.

The simulation reveals that the simulated mosquito combat strategy is unable to cope with the increase in the number of mosquitos. This leads to memory scalability issues for various simulations, as the simulation progresses and the number of individual elements, such as mosquitos and eggs, increase significantly. Thus, memory constraints limit the number of days that can be simulated according to the approach's memory usage. Table II shows the maximum number of elements that we are able to simulate with each approach. Note that if the simulated mosquito combat strategy were adequate, the number of individual elements in the simulation would be kept under control, and the scalability issues would not arise. So, despite the scalability constraints, we consider three of the examined realizations to be feasible (Java, Scene and RePast). The performance of the realization in GROOVE deserves some special attention. The relatively poor performance can be explained by the fact that the tool is originally aimed at exhaustive exploration of a state space. Because of that, the execution of the simulation in this tool differs from the execution in all the other approaches, which take benefit from the execution of a single path is the whole state space. (For example, while in the three other approaches a mosquito flies to only one house within range, in the GROOVE simulation, a new state of the simulation is produced for each possible migration departing from a particular house.)

TABLE I
Execution Time in Milliseconds

Days	*Java*	*GROOVE*	*Scene*	*RePast*
5	22	48,224	5,783	49,038
10	28	861,846	8,311	92,073
15	33	–	9,854	122,607
20	40	–	11,288	146,686
25	76	–	107,747	1,039,921
30	115	–	488,994	3,445,092
35	206	–	957,218	6,642,166
40	254	–	1,782,615	10,276,445
50	822	–	–	–
60	7,235	–	–	–
70	330,112	–	–	–
80	33,757,062	–	–	–

TABLE II
Maximum Number of Elements in Simulation

	Days	**Mosquitos**	**Eggs**
Java	82	37,118,209	232,602,309
Groove	10	665	8,097
Scene	42	78,129	309,638
RePast	43	79,359	317,728

V. Related Work

Most approaches for simulation in the situation awareness literature, such as [2], [3], [24], [25] address simulation from the perspective of humans in the loop. Their objective is mainly to examine observe human performance and behavior under controlled conditions. Here, instead, we have focused on fully-automated simulations as a means to gain insight into the consequences of particular action strategies. Differently from the works cited, the behavior of humans in the setting we have considered is quite restricted in scope, and limits itself to clear cut mosquito fighting actions of the healthcare professionals. Considering more sophisticated behavior for the healthcare professionals (and perhaps of the overall population) would be a natural extension of this work. This is because the combination of community-based programs with chemical control of the mosquito has yielded significant results [26] apud [27]. As discussed in [28], the success of community-based strategies depends, among other things, on the behavior of the people, and strategies involved.

Concerning the application area itself, an advanced effort in the simulation of dengue fever is [1]. The authors have proposed a tool to assist the design of (site-specific and vector population-specific) control strategies for dengue. They have provided, among other elements: (i) a collection of built-in models that can be combined to represent various scenarios; (ii) a modeling language; and (iii) a graphical interface for model selection and configuration. Models are implemented using the high-level programming language TerraML (Terra Modeling Language) from TerraME [4], which is based on the Lua programming language [29].

TerraME combines multiple paradigms such as agent-based simulation, cellular automata abstractions and scheduled events [4]. Although it also supports scheduled events, the simulation progresses through well-defined steps, by invoking code (functions such as execute and synchronize) that must be supplied by the designer of the simulation. We believe that the TerraME framework could be made even more comprehensive with the support for rule-based situation detection and complex rule-based event processing.

VI. Conclusions and Future Work

In this paper, we have worked out a simulation scenario in four different realization approaches. The approaches have been selected because of their complementary characteristics. The object-oriented programming approach was selected as a representative of a mainstream implementation approach. It has offered the best time performance for simulation execution in the experiment conducted. Nevertheless, from a design perspective, we can observe that it offers no support for situation management or other high-level abstractions for simulation. The graph-based approach was selected because of its radical departure from the conventional approach. It incorporates a model-based front-end with fully declarative transformation rules. Despite the benefits in the visual specification of situation elements and high-level rules, we have observed that the graph-based approach was not able to deliver the time and memory performance to address the simulation scenario. This can be explained by the fact that the tool is originally aimed at exhaustive exploration of a state space as discussed in the results section. The agent-based approach in its turn, featured a programming style that is quite similar to the object-oriented programming approach. Despite that, it was supported by an easy to use graphical interface, directly connected to simulation visualization. Spatial aspects were also supported directly in this approach favoring the specification simulations of physical environments. Finally, the rule-based situation-aware approach was selected because of the high-level abstractions employed. We have observed, as expected, that the declarative approach favors a specification that reflects domain rules more directly. Concerning the performance aspects, we have observed that the declarative specification and high-level interpretation of rule sets by the rule engine has not constituted a hurdle for the scalability of the approach, and we were able to simulate a large number of elements in the proposed case study (over 380 thousand elements, such as mosquitos and batches of eggs).

We conclude that an ideal solution would require the combination of a model-based front-end with the abstraction provided by the declarative specification of rules. We believe this can be pursued with a combination of the existing approaches with a platform such as Scene/Drools Fusion. This perhaps can be obtained with the use of a generative model-transformation based approach (such as the one some of us employed in [30]). Implementing such a vision is the scope of our ongoing efforts.

Since we have observed that a situation-aware platform such as Scene [7] can be used for simulation but also for efficient runtime execution, we believe this can serve as a design strategy. In this case, situation specification and detection can be first explored in simulation and then be put to work in real-time. We are currently exploring this strategy in the scope of environmental disaster management, in particular in a project concerning water pollution in the Rio Doce basin after the massive rupture of an iron ore tailings dam. One of the challenges we intend to address concerns the integration of (water quality) historic records and sensor data into a situation-aware solution.

Acknowledgments

This work is partly supported by CNPq (407235/2017-5 and 312123/2017-5) and CAPES (23038.028816/2016-41).

References

[1] T. F. M. de Lima, R. M. Lana, T. G. de Senna Carneiro, C. T. Codeço, G. S. Machado, L. S. Ferreira, L. C. Medeiros, and C. A. Davis Jr, “Dengueme: A tool for the modeling and simulation of dengue spatiotemporal dynamics,” *International Journal of Environmental Research and Public Health*, vol. 13, no. 9, 2016.

[2] V. F. Mancuso, D. Minotra, N. Giacobe, M. McNeese, and M. Tyworth, “Idsnets: An experimental platform to study situation awareness for intrusion detection analysts,” in *Cognitive Methods in Situation Awareness and Decision Support (CogSIMA), 2012 IEEE International Multi-Disciplinary Conference on*, IEEE, 2012, pp. 73–79.

[3] A. Angelopoulou, K. Mykoniatis, and W. Karwowski, “A framework for simulation-based task analysis-the development of a universal task analysis simulation model,” in *Cognitive Methods in Situation Awareness and Decision Support (CogSIMA), 2015 IEEE International Inter-Disciplinary Conference on*, IEEE, 2015, pp. 77–81.

[4] T. G. de Senna Carneiro, P. R. de Andrade, G. Câmara, A. M. V. Monteiro, and R. R. Pereira, “An extensible toolbox for modeling nature–society interactions,” *Environmental Modelling & Software*, vol. 46, pp. 104–117, 2013.

[5] N. Collier, “Repast: An extensible framework for agent simulation,” *The University of Chicago’s Social Science Research*, vol. 36, p. 2003, 2003.

[6] P. D. Costa, G. Guizzardi, J. P. A. Almeida, L. F. Pires, and M. van Sinderen, “Situations in conceptual modeling of context,” in *Enterprise Distributed Object Computing Conf. Workshops, 2006. EDOCW’06. 10th IEEE Int.*, IEEE, 2006, pp. 6–16.

[7] I. S. Pereira, P. D. Costa, and J. P. A. Almeida, “A rule-based platform for situation management,” in *Cognitive Methods in Situation Awareness and Decision Support (CogSIMA), 2013 IEEE International Multi-Disciplinary Conf. on*, IEEE, 2013, pp. 83–90.

[8] A. H. Ghamarian, M. de Mol, A. Rensink, E. Zambon, and M. Zimakova, “Modelling and analysis using GROOVE,” *Int. Journal on Software Tools for Technology Transfer*, vol. 14, no. 1, pp. 15–40, 2012.

[9] C. Lok, N. Kiat, and T. Koh, “An autocidal ovitrap for the control and possible eradication of aedes aegypti.,” *The Southeast Asian J. Tropical Medicine and Public Health*, vol. 8, no. 1, pp. 56–62, 1977.

[10] V. Morato *et al.*, “Infestation of aedes aegypti estimated by oviposition traps in brazil,” *Revista de Saúde Pública*, vol. 39, no. 4, pp. 553–558, 2005.

[11] P. F. C. Vasconcelos, “Doença pelo vírus zika: Um novo problema emergente nas américas?” *Revista Pan-Amazônica de Saúde*, vol. 6, no. 2, 2015.

[12] I. A. Braga and D. Valle, “Aedes aegypti: Histórico do controle no Brasil,” *Epidemiologia e serviços de saúde*, vol. 16, no. 2, pp. 113–118, 2007.

[13] N. Sivagnaname, K. Gunasekaran, *et al.*, “Need for an efficient adult trap for the surveillance of dengue vectors,” *Indian Journal of Medical Research*, vol. 136, no. 5, p. 739, 2012.

[14] A. M. Baldi, E. Zambon, P. D. Costa, and E. M. Montiel, “Simulação de aplicação de armadilhas no combate ao aedes aegypti,” in *XVII Workshop de Informática Médica - São Paulo, SP*, 2017.

[15] H. A. Maimusa, A. H. Ahmad, N. F. A. Kassim, and J. Rahim, “Age-stage, two-sex life table characteristics of aedes albopictus and aedes aegypti in Penang Island, Malaysia,” *Journal of the American Mosquito Control Association*, vol. 32, no. 1, pp. 1–11, 2016.

[16] World Health Organization, *Dengue Control*, http://www.who.int/denguecontrol/research, 2017.

[17] F. Dzul-Manzanilla *et al.*, “Indoor resting behavior of aedes aegypti (diptera culicidae) in Acapulco, Mexico,” *Journal of Medical Entomology*, 2016.

[18] OpenStreetMap, *OpenStreetMap*, http://www.openstreetmap.org, 2017.

[19] M. P. C. Piccin, “Avaliação da relação da densidade de vetores e da presença de Aedes aegypti infectados com a ocorrência de dengue na cidade de Vitória,” Master’s thesis, Fed. Univ. of Espírito Santo, 2013.

[20] R. M. Freitas, A. E. Eiras, and R. L. Oliveira, “Calculating the survival rate and estimated population density of gravid Aedes aegypti (Diptera, Culicidae) in Rio de Janeiro, Brazil,” *Cadernos de Saúde Pública*, vol. 24, pp. 2747–2754, Dec. 2008.

[21] A. Baldi, *GitHub Repository*, https://github.com/alexnede?tab=repositories, 2017.

[22] A. Rensink and E. Zambon, *GROOVE - GRaphs for Object-Oriented VErification*, http://groove.cs.utwente.nl, 2017.

[23] P. Browne, *Jboss drools business rules*. Packt Publishing Ltd, 2009.

[24] E. Özyurt, B. Döring, and F. Flemisch, “Simulation-based development of a cognitive assistance system for navy ships,” in *Cognitive Methods in Situation Awareness and Decision Support (CogSIMA), 2013 IEEE International Multi-Disciplinary Conference on*, IEEE, 2013, pp. 22–29.

[25] O. Juarez-Espinosa and C. Gonzalez, “Situation awareness of commanders: A cognitive model,” 2004.

[26] A. Baly, M. Toledo, M. Boelaert, A. Reyes, V. Vanlerberghe, E. Ceballos, M. Carvajal, R. Maso, M. La Rosa, O. Denis, *et al.*, “Cost effectiveness of aedes aegypti control programmes: Participatory versus vertical,” *Transactions of the Royal Society of Tropical Medicine and Hygiene*, vol. 101, no. 6, pp. 578–586, 2007.

[27] I. A. Rather, H. A. Parray, J. B. Lone, W. K. Paek, J. Lim, V. K. Bajpai, and Y.-H. Park, “Prevention and control strategies to counter dengue virus infection,” *Frontiers in Cellular and Infection Microbiology*, vol. 7, p. 336, 2017.

[28] V. S. Nam, N. T. Yen, T. V. Phong, T. U. Ninh, *et al.*, “Elimination of dengue by community programs using Mesocyclops (Copepoda) against Aedes aegypti in central vietnam,” *The American J. of Tropical Medicine and Hygiene*, vol. 72, no. 1, 2005.

[29] R. Ierusalimschy, L. H. Figueiredo, and W. Celes Filho, “Lua-an extensible extension language,” *Softw., Pract. Exper.*, vol. 26, no. 6, 1996.

[30] P. D. Costa, I. T. Mielke, I. Pereira, and J. P. A. Almeida, “A model-driven approach to situations: Situation modeling and rule-based situation detection,” in *Enterprise Distributed Object Computing Conf. (EDOC), 2012 IEEE 16th Int.*, 2012.