Extending a Multi-Agent Systems Simulation Architecture for Systems-of-Systems Security Analysis

Jamal El Hachem, Vanea Chiprianov, Valdemar Vicente Garcia Neto, Philippe Aniorte

Extending a Multi-Agent Systems Simulation Architecture for Systems-of-Systems Security Analysis

Jamal EL HACHEM*, Vanea CHIPRIANOV*, Valdemar Vicente GRACIANO NETO†, Philippe ANIORTE*
*UNIV PAU & PAYS ADOUR, LIUPPA, PAU, FRANCE
{jamal.elhachem, vanea.chiprianov, philippe.aniorte}@univ-pau.fr
†University of São Paulo, São Carlos, Brazil, valdemarneto@usp.br

***Abstract*—Security is an important concern for software-intensive Systems-of-Systems (SoS). Architectural analysis for SoS secturity assessment should be performed at early stages of development. Such activity could prevent vulnerabilities and avoid potential cascading attack emergent behaviors, i.e., a succession of security vulnerabilities that emerge from individual constituents security fragilities, potentially causing interruption and collapse of SoS operation.**

Model simulation can prevent these issues by predicting, at design-time, how SoS will behave regarding its reaction to potential attacks. As security is a quality attribute, i.e., a property that comes up from the relation between software parts, software architecture analysis and simulation are an additional support for the prediction of SoS security. However, despite recent advances in such area, few simulation approaches have tackled simulation of secure SoS architectures where the basis of the described models are the SoS behavior or the interactions among the SoS Constituent Systems (CS).

The main contribution of this paper is offering a big picture of how recent advances on SoS security analysis via simulations can form a robust framework for SoS security prediction. We argue the pertinence of Multi-Agent Systems (MAS) for SoS simulation due to similarities between MAS and SoS concepts, and we report how MAS simulation enables the visualization of emergent behaviors and how they impact the SoS security. Our results to foster SoS security analysis include (i) an extension of a MAS conceptual model and platform to include security concepts, (ii) a Model-Driven Engineering (MDE) approach that adopts automatic mappings between secure SoS architecture modeled using an existing SysML-based modeling language, namely the SoSSecML, and (iii) a MAS platform to support such analysis.

I. Introduction

The Systems-of-Systems (SoS) concept has been used since 1950s to describe the Systems that are primarily composed of distributed, concurrent and independent systems that interact to accomplish a common goal. Maier differentiates SoS by their essential characteristics [1] referred to by the acronym "OMGEE": Operational and Managerial independence of the CSs, Geographic distribution, Evolutionary development and Emergent behavior. Several other properties could characterize SoS such as: global mission, autonomy, connectivity, heterogeneity and diversity [2]. SoS complexity arises mainly from the fact that it is difficult to analyze their behavior and predict their properties even if the properties of their CS are given [3]. These specific characteristics introduce considerable challenges into SoS engineering such as simulating and analyzing the SoS architecture to identify emergent behaviors [4][5].

Besides the analysis challenges rising from SoS complexity and specific characteristics, an additional challenge is the consideration and analysis of some important SoS quality attributes such as security, safety, adaptability and performance. In this work, we focus on SoS security. Indeed, as argued in [6][7], SoS suffers from traditional security concerns related to their complex CS, as well as additional emergent security problems arising from their specific characteristics.

Over the last years, there has been a growing awareness of SoS being exposed to severe security attacks resulting in dangerous impact on SoS services and functionalities, nation's economical plans, and more importantly the citizen's safety. Some examples of recent massive attacks targeting SoS application domains with widely-encompassing effects are the following: The Dyn Cyberattack[1] in 2016 targeting the Internet-of-Things SoS application domain; the Stuxnet attack[2] in 2010 and its successor Irongate in 2016 and the Google building attack[3] in 2013. What is common to these massive attacks is that they are usually established by single vulnerabilities initially judged as insignificant or low-impact for the systems (CS) on which they were identified, but the cascade/sequence of several triggered vulnerabilities induced by the use of the systems (CSs) in common/together (SoS interactions) intensify the attack and magnify its effect.

Many of the CSs may suffer from small, known and unresolved vulnerabilities. These known vulnerabilities could be exploited and connected in an unknown way, resulting in a *cascading attack emergent behavior*, being enabled by the CSs interactions that are necessary to achieve the SoS global goal. Some of these vulnerabilities could be hidden or unknown, but plenty of them are known and not adequately addressed. Moreover, the consequences of such attacks on the SoS cannot be understood by means of the mere evaluation of the behavior of each single CS, but require an assessment of the effect of the interactions on the behavior of the whole SoS [8][9].

[1] money.cnn.com/2016/10/22/technology/cyberattack-dyn-ddos/index.html
[2] http://large.stanford.edu/courses/2015/ph241/holloway1/
[3] https://www.wired.com/2013/05/googles-control-system-hacked/

In our work, we focus on this kind of security attacks, which has a possible extensive impact on SoS. We call this security concern a *cascading attack* and we define it as follows: *A cascading attack is an attempt to destroy, expose, alter, disable, steal or gain unauthorized access to or make unauthorized use of an asset by exploiting an unknown succession/sequence of security vulnerabilities triggered in different constituent systems, and connected through the interactions between these CSs when collaborating to accomplish the SoS global goal(s).*

The aim of this study is to address the SoS analysis challenge while considering SoS emergent behavior, CSs managerial and operational independence and security properties. We focused on the following research question: *How can secure SoS architectures be simulated and consequently analyzed, so as to discover security, emergent, unknown cascading attacks, early at the architecture phase, in order to avoid time and cost wastage of later changes and to prevent massive damages targeting the SoS functionalities and impacting people's safety and security*?

To answer this question, we performed a review of the SoS analysis approaches and argued the usefulness of Multi-Agent Systems (MAS) for SoS simulation due mainly to the similarities between MAS and SoS concepts. Afterwards, we proposed and implemented a MAS security extension to express cascading attacks related concepts in order to properly execute and analyze secure SoS architectures. Additionally, we defined and implemented a matching mechanism that helps identifying the possible concatenation/sequence of triggered vulnerabilities in order to discover emergent unknown security cascading attacks. Finally, we used Model-Driven Engineering (MDE) to implement a generator tool (a set of Model-To-Text transformation rules) to (semi)automatically map the secure SoS architectures modeled using an SysML-based modeling language, the SoSSecML [10], to the extended MAS platform.

The remainder of this paper is organized as follows: Section II discusses the related work for secure SoS analysis. Section III presents our proposed MAS security extension. In Section IV we present our model-based generator tool. Section V presents our final remarks and Section VI concludes the paper.

II. Investigating Analysis Approaches for Secure SoS Architecture Simulation

Model simulation is a powerful analysis technique for developing a level of understanding of the real behavior of a complex system by analyzing the approximate working conditions [5][11]. Several initiatives have been proposed to address SoS simulation and analysis, but only few novel studies consider SoS security. In this section, we review the analysis approaches for secure SoS and we evaluate them regarding their ability to analyze SoS taking into account SoS OMGEE characteristics as well as their ability to achieve security analysis. Subsequently, we examine the strengths and limitations of each studied approach in addressing the emergent behavior, in particular the emergent cascading attacks resulting from the SoS interactions. Moreover, even if the number of these studies is limited and their basis are diverse, we regrouped them, based on common features, in the following sub-categories: 1) Approaches based on threat analysis processes; 2) Graph-based approaches and 3) Goal-oriented/Agent-based approaches.

A. *Approaches based on threat analysis processes*

Kobetski and Axelsson investigate in [12] the challenges towards safe and secure SoS. They propose to use the system theoretic safety analysis method to systematically view possible undesired losses in SoS. The method consists in 1) identifying a set of possible undesirable losses (accidents) and system hazards that may lead to an accident; 2) deducing a set of safety requirements; 3) identifying a control structure to avoid the hazardous states in the SoS; 4) exploring the control actions again to identify the situations that may lead to hazards. This recent work discusses general approaches to undertake each step without detailing the methods, techniques and tools to be used; neither detailed guidance on how the possible losses and control structures should be identified and defined. In addition, the authors recognize that their method needs further development to be applicable to SoS mainly due to the managerial and operational characteristics. Moreover, they mention the applicability of their approach to analyze the security of SoS without arguing this point.

Mori et al. [13] propose a framework for evolutionary SoS threat analysis. They apply the formal concepts analysis technique to study the impact of adding or removing assets in an SoS scenario in terms of threats and/or vulnerabilities that can arise from this evolution. The analysis algorithm takes as input a scenario affected by the structural and emerging threats (statically identified), their corresponding mitigation strategies and the scenario to achieve. It gives as output the distribution of the threats across the scenarios. The proposed approach for runtime analysis mainly considers the evolutionary characteristic of SoS. However, the evolution in the SoS scenario is enacted by the SoS administrator, thus there is a need of centralized authority. The authors do not explain clearly how the managerial and operational independence of the SoS are considered when defining and affecting the threats. Moreover, the approach analyzes the distribution of known threats/vulnerabilities over the CSs at runtime without considering all the defined vulnerabilities, not only those triggered and linked due to the interactions among the CSs.

B. *Graph-based approaches*

Nicklas et al. in [14] propose an approach, based on use-cases scenarios, that consider the safety and security aspects in smart home applications. The approach includes four steps: 1) use cases description to specify a virtual SoS composed of many cyber physical systems; 2) manual derivation of risks related to the SoS use cases, and suggestion of safety use cases with specific security measures to overcome these risks; 3) security assessment of the SoS use cases using attack trees, based on attack scenarios; 4) integration of safety use cases to harmonize the security structure to the safety requirements. This approach seams promising for assessing the risks related

to a specific SoS application domain and based on defined attack scenarios. However, the idea lacks deeper investigation specially to propose proper methods for each step as well as to implement proper tools.

In [8], Guariniello et al. modified the Functional Dependency Network Analysis to make it applicable for SoS to analyze the internal and external impact of cyber attacks on the SoS interdependencies. To perform the analysis, the SoS is represented as a directed network with nodes to represent the CSs or their capability and links to represent the dependencies with two values indicating the strength and criticality of each dependency. Then, based on a set of formulas, the result is a percentage indicating the degradation in the CS operability. In this study, the security is barely considered by adding a weight indicating the availability of data to model the effect of an attack. This value could be defined by an expert or through data simulation as mentioned by the authors. There are no security concepts describing the vulnerabilities or the attack, neither the SoS emergent behavior.

Similar to [8], Dahmann et al. [15] propose a general framework to assess the security risk of a single security incident on multiple constituent systems. The authors propose the use of the Functional Dependency Network Analysis to measure the operational effectiveness of the missions in the SoS. The analysis intended to assess the ability of the SoS to operate effectively if one or more CS fail.

C. *Goal-oriented/Agent-based approaches*

Abercrombie and Sheldon in [16] use the game theory implemented in dynamic Agent Based Game Theoretic simulations to analyze the information security of smart grids. Two players game, an attacker (any hacker)-defender (SoS administrator) interaction scenario is described using agents to assess the risk in an SoS in terms of the probability of a successful attack. Even if this approach seems promising to calculate the cumulative probability of successful attacks at a specific time of a scenario simulation, the attacker/defender model, in its current form, is too simple to represent an SoS with its CSs and their interactions. Moreover, the analysis does not reveal specific information on the vulnerabilities and how they are triggered, leading to the successful elementary attacks.

Meland et al. introduce in [17] an approach to analyze the threats at the requirement level of the SoS development. They extend an agent-based modeling language, with threat analysis. The extension consists in an algorithm to calculate how the impact of a threatening event is propagated in the model. This analysis is based on a set of identified threatening event and assumptions/propagation rules over goal decomposition trees which make it limited to the specified rules. Moreover, the authors do not detail how these rules were selected and defined, neither why they are suitable for the air traffic management domain (which could be considered an SoS). In addition, the proposed analysis process ends when all the elements are visited, which expose it to the combinatorial explosion issue in the context of SoS.

D. *Discussion*

The general analysis of the identified categories could be summarized as follows:

1) The approaches based on threat analysis processes [12][13] only focus on static scenarios to analyze the distribution of threats and hazardous states. They lack guidance and methodologies to support the unknown emergent behavior and do not consider the interactions and security details that trigger the threats and attacks;
2) Graph-based approaches [8][14][15] suffer from issues related to the combinatorial explosion, specially in the context of complex SoS where there is a high number of possibilities and interactions;
3) Goal-oriented/Agent-based approaches [16][17] seem to be promising to analyze the SoS interactions and related security issues through simulations.

A deeper conclusion about the studied approaches is that many of them touch on security only in a broad way. Most of the results are restrained to the effects in term of impact on the operability of the existing SoS operations. The majority of the studied approaches do not explicitly or clearly consider the OMGEE SoS specific characteristics in the analysis. Moreover, they lack pertinent details to analyze SoS interactions and emergent behavior arising from CSs operations and interactions without a centralized overview of the SoS (CSs operational and managerial independence), as well as to support the early discovery and anticipation of unforeseen security problems.

III. Proposal: Multi-Agent Systems Extension for SoS Secure Architecture Analysis and Cascading Attacks Discovery

Agent-based simulation approaches were extensively used with the intention to execute the architectural choices and understand the impact of an individual system behavior on the global behavior. For that such approaches form a solid basis when exploring the CS and SoS behaviors [18]. By performing agent-based simulation, the SoS behavior could be understood and analyzed to enable the improvement of the SoS architectures (structure and behavior/functionalities) while maintaining control and minimizing vulnerability of the SoS.

Among the agent-based approaches, Multi-Agent Systems (MAS) simulation seems the most suited and actively used technique to express SoS aspects. Indeed, an agent is any autonomous entity that plays certain roles and interacts with other agents to achieve individual or common goals [19]. MAS are distributed intelligent systems composed of agents that cooperate and coordinate to solve their local goals as well as global goals. Therefore, they provide the means to represent the SoS architectures if the concepts of the models are faithfully mapped to the agent-based concepts. In the following sub-sections, we will present MAS and argue their usefulness for SoS simulation.

A. MAS Conceptual Model

Figure 1 presents the extension that we performed based on the MAS specifications[4] to include security aspects for SoS simulation modeling. We represent in the figure - concepts in black and white - a part of the MAS conceptual model that we designed using Ecore[5]. For this conceptual model we focused on the essential MAS concepts that might be used and/or extended to express the SoS concept semantics and therefore allow a proper simulation and analysis of secure SoS architectures, mainly *Agent, Behavior* and *ACLMessage*.

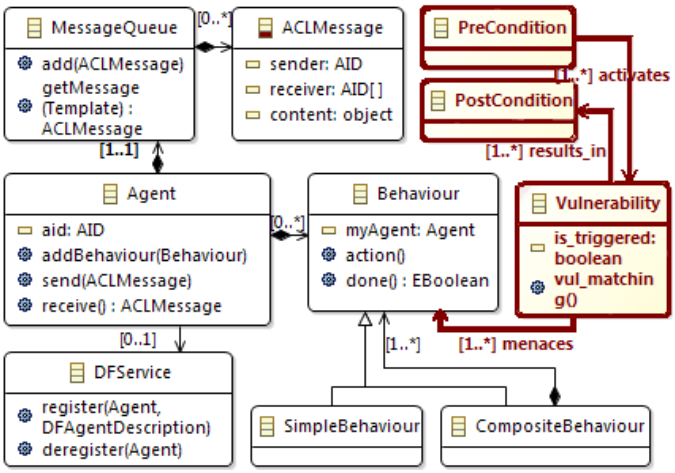

Fig. 1. Part of the MAS conceptual model with our security extension

Actually, in MAS, the *Agents* are behavior-oriented, their behaviors are created by extending the *Behavior* class and adding it to a behavior pool to be then executed concurrently based on a scheduler/execution model. A behavior can be *simple* or *composite* in which case is formed by a combination of simple behaviors.

The communication between agents is done by the exchange of asynchronous messages. The format of those messages is defined by the Agent Communication Language (ACL), conforming to the FIPA standardization, and they are implemented as objects of the *ACLMessage* class. Each agent has a *MessageQueue* where the messages coming from other agents are stocked.

All agent-level operations, such as sending messages, moving and even starting and terminating, are defined by a service (such as the service helper). The agent and its services information are registered in the *DFService* (Domain Facilitator).

B. Applying Agent-based analysis to Systems-of-Systems: Alignment of concepts between SoS and MAS

To perform useful analyses of the secure SoS emergent behavior arising from CS interactions, the analysis approach should provide a solid alignment between the SoS modeling domain and the simulation domain where the models will be executed and analyzed. Indeed, the analysis of the architectures conceived using any modeling language should underly the semantics of the model and its specification features [20]. The execution (executable artifacts) of the defined architectures should hold the properties described in the models [21].

Consequently, we performed an analysis of SoS and MAS domains, an we obtained a number of similarities between their artifacts. Similar to CSs of an SoS, agents in MAS are distributed, independent, autonomous and evolutionary:

- Local views: No agent has a full global view of the system;
- Decentralization: There is no designated controlling agent;
- Self organization: The overall orders in MAS arise from the interactions between the agents;
- Large problem solving: MAS intend to solve realistic and large scale problems that are beyond the capabilities of an individual agent;

Moreover, MAS are suitable approaches to express the SoS OMGEE specific characteristics, as described below:

- CSs operational and managerial independence could be expressed by the fact that agents in the MAS have their own goals. Like a CS, no agent has a global vision of the system;
- SoS geographical distribution could be translated into distributed agents;
- SoS evolutionary development could be denoted by the flexibility, evolution and openness of MAS systems;
- Emergent behavior could be expressed by agents autonomy and pro-activity.
- SoS context could be represented by agents environment.

Accordingly, we aligned the SoS concepts with the corresponding MAS concepts based on the similarities between them allowing by that an adequate alignment between both SoS and MAS domains. Table I exhibits the essential similarities/alignments between the SoS and MAS concepts.

1) Constituent System to Agent: This first alignment is between the CS concept of the SoS domain and the *Agent* concept of the MAS domain.

Indeed, a CS is an operationally independent software system that is intended to achieve a complex task. It operates independently to realize its own goals - what is called *individual* goals in SoS and agent paradigms, and it interacts cooperatively with other CSs to achieve a higher goal that none of the CSs is able to accomplish in isolation (*global* goal). In addition, a CS is managerially independent because different CSs could belong to different organizations and they are managed by those organizations. Moreover, due to this managerial independence, none of the CSs has a global view of the whole SoS composition and interactions (local views).

Likewise, an agent is an autonomous software system that acts following its own list of behaviors or upon its perception of its environment what is called operational independence in the SoS paradigm. In MAS, the agents are self-organized, in addition they could be classified in different structural dimen-

[4]MAS is defined by the the Foundation for Intelligent Agents (FIPA: http://www.fipa.org/repository/standardspecs.html)

[5]http://www.eclipse.org/modeling/emf/

TABLE I
ESSENTIAL SIMILARITIES BETWEEN SOS AND MAS CONCEPTS

SoS concept	MAS concept	Similarities
SoS	MAS	Distributed systems composed of multiple independent/autonomous systems that interact/communicate to accomplish individual/local goals and global/collaborative goals.
Constituent System	Agent	Operational independence/autonomy ; individual-global operations/local-collaborative behaviors; managerial independence/local views.
Functionality/operation	Behavior/Task	Simple-complex operations/simple-composite behaviors; sequential and parallel/concurrent operations/behaviors.
CS interactions	ACL Messages	Express interactions/communications between the operations/behaviors defined on CS/Agent interfaces.

sions/organizations that can be distributed over the network (managerial independence).

2) Functionality to Behavior: Another correspondence is between the functionality/operation concept of the SoS domain and the *behavior/task* concept of the MAS domain.

Actually, a CS operation could describe the CS mission or global functionality representing the operation on the CS interface with witch it participates in the realization of the SoS global goal. Furthermore, a CS can participate in the achievement of several individual goals at the same time, accordingly it can interact with several other CSs and execute several operations in parallel.

Similarly, a behavior in MAS describes a task that an agent can accomplish. These behaviors can be simple (such as TickerBehavior, OnShotBehavior, CyclicBehavior) or composite (such as ParallelBehavior, SequentialBehavior) to create complex tasks by composing simple behaviors. An agent can execute several behaviors concurrently.

3) Interactions and ACL Messages: Another similarity is between CS interactions and agents interactions.

In the SoS, the CSs are dispersed and they exclusively communicate information (geographic distribution specific characteristic). With respect to this idea, the interactions between different operations defined on different CS interfaces are expressed through the *interaction* concept.

Likewise, the agent communications in MAS are achieved through the exchange of *ACL messages*. Indeed, the agents interact with other agents belonging to the same platform or to different agent containers/platforms by way of ACL messages and message transport services. The structure of the messages is a set of key values written in FIPA-ACL. The content of the message is expressed in a content language, and content expressions can be grounded by referenced ontologies.

C. MAS Conceptual Model Security Extension

As stated in section II, MAS is an appropriate choice for SoS simulation as previously argued, although MAS lack concepts to express SoS security aspects.

However, to express the dynamics and execute the whole behavior of secure SoS architectures in such a way as to enable some types of emergent behavior to manifest and therefore become analyzable (in particular the unknown security cascading attack), it is crucial to express the semantics of the secure SoS architectures security concepts. Therefore, it is necessary to assign executable MAS notations to the SoS security attack concepts specified in the SoS architectures.

Nonetheless, existing MAS approaches lack such concepts. In the past years many attempts have been made to include security into MAS. However, the following recent surveys [22][23][24] show that most of the proposed approaches focus on securing the agent communications, using techniques such as cryptographic signature, message encryption and access control to facilitate agent authentication and authorization.

Only few researchers propose deeper security approaches for MAS development. Mouratidis et al. [25] and Beydoun and Low in [26] propose a security-aware general approach for MAS consisting in a framework independent MAS conceptual model. Despite its potential in representing MAS security requirements, the proposed approach remains at the conceptual level without a MAS platform and tools to support its usage.

For these reasons, we propose an extension of the MAS conceptual model to express SoS security attack related concepts. Our extension (see figure 1 - bold concepts in red) mainly covers the vulnerability, pre-condition and post-condition concepts detailed in what follows:

- The concept *Vulnerability* represents a back door or weakness that *menaces* an agent behavior. For each behavior we can assign one or many vulnerabilities. A vulnerability *is_triggered* by a matching mechanism. This matching mechanism matches the pre-conditions of a vulnerability "i" in the possible cascade with the post-conditions of the previous vulnerabilities in that cascade, triggered through the interactions between the agents, allowing by that the discovery of the possible sequence of vulnerabilities that might be triggered due to the agent interactions.
- *PreCondition* defines the condition which *activates*/triggers a vulnerability. A vulnerability may have one or many pre-conditions, and a pre-condition is assigned to one and only one vulnerability via the relationship *activates*;
- *PostCondition* defines the condition which results from an activated/triggered vulnerability. A vulnerability may *results_in* one or many post-conditions, and a post condition is related to one and only one vulnerability.

Our extension of the MAS MetaModel refines the general security-aware framework-independent MAS MetaModel proposed by Beydoun [26]. It could be re-used and further extended to model and analyze other security concepts such

as security requirements and mechanisms.

IV. Automatic Mappings from secure SoS Architectures to MAS Simulation

To show the applicability of the MAS security extension that we are proposing, we implemented this extension in a well-known MAS platform to allow the execution/simulation of secure SoS architectures and the discovery of the possible related cascading attacks.

To further benefit from the execution/simulation, it is fundamental to define an easy mapping between the modeling and the execution phases to modify/adjust the secure SoS architecture and re-analyze it until reaching an acceptable level of security. These iterations allow continual refinement of the secure SoS architectures, addressing therefore the SoS security problems early at the architectural phase of the development life cycle to save cost, development time, and protect the SoS from high impact attacks.

To ensure this iterative aspect, we selected the Systems-of-Systems Security Modeling Language (SoSSecML) existing modeling language for secure SoS architectures modeling, and we took advantage of Model Driven Engineering (MDE) mechanisms, in particular the semi-automatic transformation mechanism, to define and implement a model-based code generator tool - a set of Model-To-Text transformation rules - to automatically generate executable MAS artifacts (that could be simulated in our extended MAS platform) from secure SoS architectures designed using SoSSecML.

A. *Implementing the MAS security extension in the Java Agent DEvelopment Framework (JADE)*

A good collection of simulation frameworks/platforms for the execution of MAS can be found in [27][28]. We adopted the JADE simulation platform to implement our MAS security extension because JADE has been classified as the most popular and the leading open source MAS framework, widely used and actively maintained [27][19].

Therefore, to implement our MAS security extensions, we extended the JADE simulator by defining the following Java classes:

- *public abstract class Vulnerability*: this class defines a list of vulnerabilities. Each vulnerability is linked to an agent behavior and possesses an "is_triggered" boolean value (false by default) and a *vul_matching* mechanism. This result is reported in the JADE *log_file*;
- *public abstract class PreCondition<L>*: to specify the list of pre-conditions that activates each vulnerability. <L> represents a generic type therefore we can model the conditions as different types, or implement in the future another (hierarchy of) classes to express these conditions;
- *public abstract class PostCondition<L>*: the list of post-conditions, the results of a triggered vulnerability;
- *public boolean vul_matching*: this mechanism identifies and logs "if" and "when" a vulnerability *is triggered* based on a matching between the pre- and post-conditions. This matching allows the discovery of the possible sequence of vulnerabilities (the cascading attacks) that might be triggered due to the agent interactions. It takes as input a *list of post-conditions* and returns as output a "true" or "false" value of the boolean variable *is_triggered*. Indeed, for each vulnerability, for each pre-condition assigned to this vulnerability, it tests if this pre-condition matches any of the post-conditions received in input - the post conditions of the triggered vulnerabilities related to previously executed operations. Once there is a match, *is_triggered* is set to "true".
  The importance of this mechanism resides in the fact that it discovers those cascading attacks resulting only from the execution of CSs operations, avoiding in this way the combinatorial explosion of existing attacks analysis approaches.

B. *SoSSecML for Secure SoS Architectures Modeling*

To exhibit the proper SoS semantics and improve the simulation results, there is a need for well defined specifications of the secure SoS architecture. For this purpose, SoSSecML is a novel modeling language that has been defined by [29] to support the specification of secure SoS architectures. SoSSecML extends the System Modeling Language (SysML) with structural and behavioral concepts and relationships specialized for SoS such as constituent system, operation and organization, and others particular for security such as vulnerability, pre-conditions, post-conditions and security mechanism. Consequently, SoSSecML serves as an appropriate modeling language choice to design secure SoS architectures. Having those architectures, we can perform our MAS security analysis, because SoSSecML specifies the essential concepts needed for our analysis.

The main concepts of SoSSecML are:

- The *Constituent System* concept that represents the CSs as "black boxes" with several *operations/action* on their interfaces, serving to model the interactions with other CSs.
- The *Organization* concept to denote the organization to which each CS belongs, illustrating by that the managerial and operational independence of the CSs;
- The *Action* concept and *Control Flow* relationship to represent the CS operations and interactions.
- Security concepts to support ulterior analysis of security problems. For each operation one or several *vulnerability* with a list of *pre-conditions* and *post-conditions* are assigned using *menaces* relationship. In addition to a *Security Mechanism* that helps *preventing* a vulnerability.

In order to gain the maximum engineering value from this modeling language, it is required to have well-defined mappings between the models designed using the language and their realizations [5]. A well defined and relatively easy mapping between the modeling and execution concepts increases the likelihood that the modeling domain concepts (in our case secure SoS) will be accurately captured by the simulation domain (in our case extended MAS platform). Consequently,

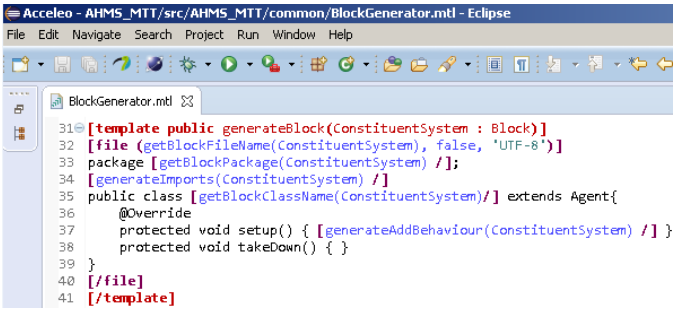

Fig. 2. CS to Agent MTT Rule

the SoS behavior and the related emergent security issues manifest through the simulation and become analyzable, allowing the discovery of the possible cascading attacks. Therefore we detail in the next subsections the mappings from SoSSecML to the extended platform.

C. *Generator tool: Transformation Rules from SoSSecML concepts to the extended MAS platform notations*

With respect to MDE, model transformations are one of the key mechanisms to define a set of rules that allow the mapping from a source model to a target model. Among the model transformation types [30], the one on which we are focusing here is the Model-To-Text (MTT) transformations (code generation). Each MTT rule allows a "vertical" transformation, from a concept of SoSSecML (high level of abstraction), towards a Java class of the extended JADE platform (implementation/simulation level of abstraction). To perform the mappings between the secure SoS and MAS concepts, we defined and applied transformation rules using Acceleo[6]: an Eclipse-based open source tool. Acceleo use a template-based approach for the automatic code generation [30]. In such approach, the mapping between source model and the produced text/code is captured through templates-rules. These rules, together with the source model are given as input to a template engine/generator which produces the code in the corresponding output language.

Figure 2 presents the first mapping rule. This rule allows the mapping from the *CS* concept of SoSSecML to the *agent* class of the extended JADE platform (*jade.core.Agent class*). As we can see in the figure, this MTT rule generates the name and the java package of the class. It sets the name and path to the class file and the import statements for the packages that are used in the class. It also generates the class body and the statements that add the behaviors to the agent class.

In a similar way, we implemented using Acceleo the MTT rules that ensure the following mappings from SoSSecML concepts to the extended JADE classes: the CS functionalities/operations are mapped to agents behaviors (*jade.core.behaviour.Behaviour*); the flows/interactions are mapped to*ACL Messages*; and the *vulnerability, pre and post-conditions* of SoSSecML are mapped to the *vulnerability, pre and post-conditions* classes that we introduced to the MAS platform. Moreover, since we are interested only in discovering the sequence of vulnerabilities that are triggered because of the CS interactions, this latter rule is defined in a way to map only the post-conditions of the *triggered* vulnerabilities linked to the *executed* operations.

Therefore, by giving these MTT rules, together with the secure SoS architecture architectures (defined using SoSSecML) as input to the Acceleo generator, the produced Java classes could be directly executed in the extended JADE platform. The simulation/execution results are logged to allow a system-level interpretation/analysis of the secure SoS architectures modeled at the abstract level.

The JADE platform includes a default specific agent, called *sniffer agent* used for logging or simply documenting the conversations between agents. However, for a better analysis, we sought a more advanced logging tool to offer a personalized log file, allowing an effortless and more efficient analysis of the simulation results. Accordingly, we integrated to JADE a custom logger that implements the open source *Log4J library*[7].

Having the custom log file issued from the simulation of the secure SoS architecture will reveal the triggered vulnerabilities, the pre-post conditions matching that triggered the vulnerabilities, as well as the CSs and interactions to which these vulnerabilities belong. Having these information captured in the log files, the security experts will be able to analyze the real behavior of the described architectures, as well as to interpret the discovered emergent sequence of vulnerabilities triggered and connected through the CS interactions and in consequence of validated pre-conditions.

V. Final Remarks

As shown in this paper, the proposed MAS conceptual model security extension and the corresponding tools is a first step towards answering our initial RQ. The use of MAS simulation to execute the secure SoS architectures allows the simulation of the secure SoS behavior. Moreover, the matching mechanism that we defined and implemented, leads to significant results in terms of discovery of high impact unknown cascading attacks arising from the succession of known unresolved vulnerabilities in the independent CSs, that could be connected due to the CSs interactions to achieve the SoS global goal. The importance of this mechanism resides in the discovery of the attacks arising only from the enabled vulnerabilities related to the executed operations, avoiding by that the combinatorial explosion that results from the analysis of all possible attack paths.

The proposed MAS security analysis approach handles the *managerial* and *operational independence* characteristics of an SoS by simulating autonomous CS/agents and analyzing their interactions needed for the realization of the SoS/MAS global goal. It also respects the *geographic distribution* characteristic since JADE allows the distributed simulation for the execution of the models. Moreover, the proposed approach addresses the *emergent behavior* of SoS through the analysis

[6]http://www.eclipse.org/acceleo/

[7]https://logging.apache.org/log4j/1.2/download.html

of the cascading attack security emergent behavior. Also, our approach partially covers the *evolutionary development* of the SoS since, by virtue of the iterative automatic mappings that we defined, it allows an easy mapping between the modeling and analysis phases, to analyze several architecture alternatives until reaching an acceptable level of security.

Our work is innovative, however, it is imperative to mention that our approach has certain limitations. Firstly, we do not fully address the emergent behavior characteristic. Being a first step in considering SoS security problems, we focused on one kind of emergence resulting from the *known* interactions and vulnerabilities. We plan to perform further investigations to cover the emergence resulting from *unknown* interactions and vulnerabilities. Secondly, in this work we defined the mappings between one specific existing language (SoSSecML) and our MAS security extended platform. It is maybe worth to generalize the mappings for several other languages and consider other security properties. Thirdly, our approach and tools need to be validated by a case study and industrial use.

VI. Conclusion

In this paper we addressed the SoS security analysis challenge, mainly the *cascading attack security* issue which has a high impact in the context of SoS. To prevent these attacks early and avoid time and cost wastage, we argued in details the usefulness of Multi-Agent Systems (MAS) for SoS simulation due to the similarities between MAS and SoS concepts. Moreover, we extended MAS with security concepts to express the semantics of cascading attacks related concepts, and we implemented these extensions in the JADE open source platform compatible with the MAS standard (FIPA). Additionally, we defined and implemented a vulnerability matching mechanism and a custom log file issued from the simulation of the secure SoS architecture to capture and reveal information needed to predict and analyze the possible cascades of triggered vulnerabilities (sequences of vulnerabilities triggered and connected through the CS interactions, in consequence of one or many successful matching and validated pre-conditions). Finally, we completed our contribution by a generator tool allowing an iterative automatic mapping between the secure SoS architectures modeling and analysis phases.

References

[1] M. Maier, "Architecting principles for SoS," *Systems Engineering*, vol. 1, pp. 267 – 284, 1998.

[2] J. Axelsson, "A systematic mapping of the research literature on system-of-systems engineering," in *10th System of Systems Engineering Conference (SoSE), Texas, USA*, 2015, pp. 18–23.

[3] F. Oquendo, "Architecturally describing the emergent behavior of software-intensive system-of-systems with sosadl," in *2017 12th System of Systems Engineering Conference (SoSE),Hawaii, USA*, 2017, pp. 1–6.

[4] J. Dahmann, "System of systems pain points," *INCOSE International Symposium*, vol. 24, no. 1, pp. 108–121, 2014.

[5] C. Nielsen, P. Larsen, J. Fitzgerald, J. Woodcock, and J. Peleska, "Systems of systems engineering: Basic concepts, model-based techniques, and research directions," *ACM Computing Surveys*, vol. 48, no. 2, 2015.

[6] J. ElHachem, "Towards Model Driven Architecture and Analysis of SoS Access Control," in *International Conference on Software Engineering Doctoral Symposium*, vol. 2, Florence, Italy, 2015, pp. 867–870.

[7] V. Chiprianov, L. Gallon, M. Munier, P. Aniorte, and V. Lalanne, "Challenges in Security Engineering of Systems-of-Systems," in *3ème Conférence en IngénieriE du Logiciel*, Paris, France, 2014, pp. 137–151.

[8] C. Guariniello and D. DeLaurentis, "Communications, information, and cyber security in systems-of-systems: Assessing the impact of attacks through interdependency analysis," *Procedia Computer Science*, vol. 28, pp. 720 – 727, 2014, conference on Systems Engineering Research.

[9] H. Kopetz, O. Höftberger, B. Frömel, F. Brancati, and A. Bondavalli, "Towards an understanding of emergence in Systems-of-Systems," in *10th System of Systems Engineering Conference (SoSE), Texas, USA*, 2015, pp. 214–219.

[10] J. ElHachem, Z. Pang, V. Chiprianov, A. Babar, and P. Aniorte, "Model Driven Software Security Architecture of Systems-of-Systems," in *Asia Pacific Software Engineering Conference*, Hamilton, New Zealand, 2016, pp. 89–96.

[11] G. Muller and C. Dagli, "Simulation for a coevolved system-of-systems meta-architecture," in *11th System of Systems Engineering Conference (SoSE), Kongsberg, Norway*, 2016, pp. 1–6.

[12] A. Kobetski and J. Axelsson, "Towards safe and secure systems of systems: Challenges and opportunities," in *Proceedings of the Symposium on Applied Computing*, ser. SAC, 2017, pp. 1803–1806.

[13] M. Mori, A. Ceccarelli, T. Zoppi, and A. Bondavalli, "On the impact of emergent properties on SoS security," in *11th System of Systems Engineering Conference (SoSE), Kongsberg, Norway*, 2016, pp. 1–6.

[14] J. Nicklas, M. Mamrot, P. Winzer, D. Lichte, S. Marchlewitz, and K. Wolf, "Use case based approach for an integrated consideration of safety and security aspects for smart home applications," in *11th System of Systems Engineering Conference (SoSE), Kongsberg, Norway*, 2016, pp. 1–6.

[15] J. Dahmann, G. Rebovich, M. McEvilley, and G. Turner, "Security engineering in a system of systems environment," in *Systems Conference (SysCon), 2013 IEEE Intl*, 2013.

[16] R. Abercrombie and F. Sheldon, "Security analysis of smart grid cyber physical infrastructures using game theoretic simulation," in *IEEE Symposium Series on Computational Intelligence*, 2015, pp. 455–462.

[17] P. Meland, E. Paja, E. Gjære, S. Paul, F. Dalpiaz, and P. Giorgini, "Threat analysis in goal-oriented security requirements modelling," *International Journal on Secure Software Engineering*, vol. 5, no. 2, pp. 1–19, 2014.

[18] W. Baldwin, B. Sauser, and R. Cloutier, "Simulation Approaches for SoS: Events-Based versus Agent Based Modeling," *In Computer Science*, vol. 44, pp. 363 – 372, 2015.

[19] F. Bellifemine, G. Caire, and D. Greenwood, *Developing Multi-Agent Systems with Jade*. John Wiley & Sons, 2007.

[20] N. Medvidovic and R. Taylor, "A classification and comparison framework for software architecture description languages," *IEEE Transactions on Software Engineering*, vol. 26, no. 1, pp. 70–93, 2000.

[21] *INCOSE Systems Engineering Body of Knowledge, version 1.6*. INCOSE UMS, March 2016.

[22] Y. Hedin and E. Moradian, "Security in multi-agent systems," *Proceedings of the 19th Annual Conference on Knowledge-Based and Intelligent Information & Engineering Systems, KES, Singapore*, vol. 60, pp. 1604 – 1612, 2015.

[23] O. Ogunnusi and S. Razak, "Attacks and security solutions for agent communication in multi-agent systems," *Proceedings of International Journal of Soft Computing*, vol. 10, pp. 99–109, 2015.

[24] S. Bijani and D. Robertson, "A review of attacks and security approaches in open multi-agent systems," *Artificial Intelligence Review*, vol. 42, no. 4, pp. 607–636, 2014.

[25] H. Mouratidis, M. Kolp, P. Giorgini, and S. Faulkner, "An architectural description language for secure multi-agent systems," *Web Intelligent and Agent System*, vol. 8, no. 1, pp. 99–122, 2010.

[26] G. Beydoun and G. Low, "Generic modelling of security awareness in agent based systems," *Information Sciences*, vol. 239, pp. 62 – 71, 2013.

[27] K. Kravari and N. Bassiliades, "A survey of agent platforms," *Journal of Artificial Societies and Social Simulation*, vol. 18, no. 1, p. 11, 2015.

[28] R. Bordini, L. . Braubach, M. Dastani, A. Seghrouchni, J. Gomez-Sanz, J. Leite, G. O'Hare, A. Pokahr, and R. Ricci, "A survey of programming languages and platforms for multi-agent system," vol. 30, 2006, pp. 33–44.

[29] J. E. Hachem, T. A. Khalil, V. Chiprianov, A. Babar, and P. Aniorte, "A model driven method to design and analyze secure architectures of systems-of-systems," in *22nd International Conference on Engineering of Complex Computer Systems (ICECCS 2017), Fukuoka, Japan*, pp. 166–169.

[30] T. Mens and P. Van Gorp, "A taxonomy of model transformation," *Electron. Notes Theor. Comput. Sci.*, vol. 152, pp. 125–142, 2006.