UNIVERSITÉ DE PAU ET DES PAYS DE L'ADOUR

ÉCOLE DOCTORALE DES SCIENCES EXACTES ET LEURS APPLICATIONS

A Model Driven Method to Design and Analyze Secure System-of-Systems Architectures. Application to Predict Cascading Attacks in Smart Buildings.

Prepared by

Jamal EL HACHEM

A thesis submitted in fulfillment for the degree of Doctor of Philosophy in Computer Science

Examination Committee:

M. Salah SADOU, PU, Université de Bretagne Sud, France (President)

Mme. Elisa YUMI NAKAGAWA, Prof., University of São Paulo, Brazil (Reviewer)

M. Khalil DRIRA, DR CNRS, LAAS, France (Reviewer)

Mme. Jenifer PEREZ BENEDI, Prof., Universidad Politecnica de Madrid, Spain (Examiner)

M. Philippe ANIORTE, PU, Université de Pau et des Pays de l'Adour, France (Advisor)

M. Vanea CHIPRIANOV, MC, Université de Pau et des Pays de l'Adour, France (Co-Advisor)

07 December 2017

To my loving parents Maha and Milad ...

Acknowledgements

"*La reconnaissance est la mémoire du cœur.*" *Hans Christian Andersen*.

Je voudrais dire *Merci* à plusieurs personnes qui ont contribué d'une manière ou d'une autre, directement ou indirectement, à la réussite de cette thèse. Si j'oublie quelqu'un, c'est la faute de ma mauvaise mémoire, donc veuillez m'excuser !

Je tiens tout d'abord à remercier le Conseil Départemental des Landes pour leur généreuse bourse de thèse. Sans ce financement, ce travail n'aurait pas été possible.

Je tiens aussi à exprimer ma sincère gratitude aux membres de mon Jury pour avoir bien voulu participer à l'évaluation de mes travaux de thèse. Je remercie particulièrement mes deux rapporteurs Mme Elisa NAKAGAWA et M. Khalil DRIRA pour leurs évaluations fortement appréciées.

Du fond de mon cœur, je remercie mes deux directeurs : Philippe ANIORTE et Vanea CHIPRIANOV. En général, le chemin d'une thèse est significativement long et dur, mais je peux dire que pour moi c'était un incroyable voyage heureux. Ceci est sans aucun doute le résultat du dévouement, de l'attention, de la confiance et du travail d'équipe avec mes deux directeurs.

- Philippe : Merci pour votre encouragement, confiance et conseils qui m'ont indiquée toujours la bonne direction pour aboutir à la fin de cette thèse! Merci aussi de m'avoir soutenue pour saisir toutes les opportunités qui m'ont permis de bien avancer sur mes travaux.

- Vanea : Merci pour tout ! Je ne sais pas à quoi on peut s'attendre de plus d'un encadrant, mais j'ai eu plus que j'avais prévu ou même mérité! Merci pour votre disponibilité, patience, confiance et vos conseils. Surtout merci pour toutes les opportunités que vous m'aviez assurées. Merci d'avoir discuté, argumenté, guidé et orienté sans jamais forcer ! Merci d'avoir été un leader et pas un boss. Je vous suis profondément reconnaissante pour l'exemple professionnel que vous êtes. Votre comportement m'a inspiré et m'a permis d'apprendre au quotidien comment assurer un équilibre entre professionnalisme

et amitié, entre encadrement et autonomie, entre directives et liberté de proposition. Tout ce que je pourrais dire ne serait pas suffisant pour vous remercier !

Ce travail de thèse a été effectué au sein de l'IUT des Pays de l'Adour, site de Mont-de-Marsan. Je remercie tous les enseignants, le personnel et les thésards qui y sont rattachés pour m'avoir accueillie chaleureusement pendant trois ans, j'ai vraiment passé trois belles années parmi vous! Merci à chacun de vous! Je tiens à remercier spécialement Laurent GALLON pour l'être humain qu'il est et le grand cœur qu'il a!

Je remercie aussi le directeur du Laboratoire d'Informatique de l'UPPA (LIUPPA), Prof. Richard CHBEIR et tous les enseignants-chercheurs de l'équipe, qui m'ont aidée pendant l'élaboration de ma thèse.

Je réserve mes derniers remerciements à mes familles :

- Merci à ma grande famille Libanaise : mes proches notamment des deux familles EL HACHEM et ABOU RJEILI, mes professeurs de maternelle au St. Georges Khenchara, collège et lycée notre dame de la Délivrande et mes professeurs à l'Université Antonine. Un grand merci à tous mes amis Libanais pour leurs encouragements et soutien. Je tiens à remercier en particulier Tarek Al Khalil pour son grand appui !

- Merci à ma grande famille Française : mes amis, je ne peux pas les citer tous mais je spécifie Timothée et Thomas pour tous les moments inoubliables que nous avons passé ensemble! Ça sera IM PO SSIBLEEE d'oublier ces jolis souvenirs. Merci à mes voisins Robert, Eliette, Domi, Pierrot, Maythé, Michel et Maylis, tous les conseillers de quartiers, les membres de la Commission Animation et les membres du Conseil Municipal ! C'est grâce à vous que je me suis vite intégrée et que j'adore Mont-de-Marsan!

- Un grand merci à mes parents Français, Janine et Guy LADEVEZE! Il n'y a pas des mots qui pourraient exprimer ma reconnaissance envers votre gentillesse, surtout votre soutien pendant les moments les plus durs de cette thèse. Je vous aime!

- Je finis mes remerciements avec ma petite famille Libanaise : Merci à mes grands parents, Marie et Émile, pour m'avoir donné l'amour du monde! Merci à mon frère Marwan et ma sœur Lara pour m'avoir apporté la joie dans ma vie ! Et à mes parents Maha et Milad le plus grand MERCI ! Merci d'avoir sacrifié votre vie pour nous! Merciiii de m'avoir donné l'opportunité de faire mes études dans de très bonnes conditions et pour m'avoir soutenue, encouragée et appuyée durant toutes ces années!! De m'avoir appris de vraies valeurs et de m'avoir permis de devenir la personne que je suis aujourd'hui. Vous étiez et vous resterez toujours mes véritables modèles inspirants

dans la vie! Je ne saurai jamais vous exprimer mes sentiments, tout ce que je peux dire c'est que je vous aimeeeeee beaucoup et c'est d'ailleurs pour cette raison que cette thèse vous est dédiée !

Abstract

Systems-of-Systems (SoS) is becoming one of the major paradigm for engineering next generation solutions such as smart cities, smart buildings, health-care, emergency response and defense. Therefore, there is a growing interest in SoS, their architecture and specially their security. However, SoS differentiating characteristics, such as emergent behavior and managerial and operational independence of its constituents, may introduce specific issues that make their security modeling, simulation and analysis a critical challenge. In this thesis we investigate how Software Engineering approaches can be extended to model and analyze secure SoS solutions for discovering high impact attacks (cascading attacks) at the architecture stage. In order to achieve our objective, we propose a Model Driven Engineering method, Systems-of-Systems Security (SoSSec), that comprises: (1) a modeling description language (SoSSecML) for secure SoS modeling and an extension of Multi-Agent Systems (MAS) for secure SoS architecture analysis, (2) the corresponding tools: a graphical editor, a code generator, an extension of the Java Agent Development (JADE) MAS simulation framework, a custom logging tool, (3) an utilization process to guide the use of the SoSSec method. To illustrate our approach we conducted a case study on a real-life smart building SoS, the Adelaide University Health and Medical School (AHMS).

Table of contents

List of figures

List of tables

General Introduction

You're driving your child to his/her university for his/her first registration. You avoided congestion thanks to the traffic management application. Your car was guided to park straight into an available space in the university's smart parking. While waiting for your child to finish registration, you start reading the electronic version of the New York Times on your smart phone/tablet/smart watch/smart glasses. You come across the Australian government annual cybersecurity report[1] *revealing "that one of its national security contractors had suffered a security breach where an attacker had compromised the network of a small Australian company with contracting links to national security projects opening by that the door to the bigger attack". You surf some other related articles on security attacks, European cybersecurity initiatives, and international cybersecurity strategies and budgets. You've had enough. You look outside to the university's smart buildings with their smart energy facilities and autonomous management systems, and ask yourself what if an attacker take control of your child's place of learning?... While looking, memories took you back to the past, to your university's "naive" buildings, to the first time you used a smart phone, the first time you heard about smart buildings, smart energy, autonomous cars, smart parking, Internet-of-Things... What? like a zillion years ago?...* This is how fast the modern software systems are advancing and their complexity is growing, as well as their exposure to cyber attacks!

Context and problems statement

In our current global society, software systems intelligence and complexity have been systematically increased to deliver a greater wealth of functionalities in numerous domains, of a broad range and scale, such as defense, emergency management and response, health-care, smart cities, transportation, smart energy grids and E-commerce. These modern solutions are application domains of a specific type of systems called System-of-Systems

[1] https://www.nytimes.com/2017/10/10/world/australia/cybersecurity-data-breach.html?rref=collection%2Ftimestopic%2FComputer%20Security%20(Cybersecurity)

(SoS). Their complexity arises from the integration of various independent, evolutionary and distributed systems named Constituent Systems (CSs) that interact to achieve a higher global goal that none of the CSs is able to accomplish in isolation. One of the main challenges rising from this complexity is the uncertainty of behavior. This uncertainty results for example from the absence of fixed specifications, the coalition of new CSs and legacy CSs, the integration of widespread CSs that interact to achieve the SoS goal(s) leading to possible expected or unexpected beneficial emergent behaviors, as well as expected or unexpected detrimental emergent behaviors [70]. Moreover, even if the properties of each CS are given and well defined, engineering the whole SoS and predicting its functional and non-functional properties remains a challenging task.

It was obvious that the domain of SoS started receiving special attention from the research community as shown by the increasing number of international conferences and workshops [128] like the International Conference on Systems-of-Systems Engineering, the International Journal of Systems-of-Systems Engineering and the International Workshop on Software Engineering for Systems-of-Systems; many European projects[2] such as Architecture for Multi-criticality Agile Dependable Evolutionary Open System-of-Systems (AMADEOS[3]) [2013-2016], Design for Adaptability and Evolution in Systems-of-systems Engineering (DANSE[4]) [2010-2014], DYnamic MAnagement of physically coupled Systems-of-Systems (DYMASOS[5]) [2013-2016], COMPrehensive modeling for Advanced Systems-of-Systems (COMPASS[6]) [2010-2014]; as well as many institutions and universities like the US Naval Postgraduate School, the National Aeronautics and Space Administration-NASA and the Purdue University. Furthermore, the professional community starts joining efforts to propose new solutions that enable accurate engineering/development of such systems. Moreover, the bibliometric studies in [128][7] show an increasing number of research publications over time, showing the growing awareness of the importance of SoS engineering.

Despite all these efforts, numerous roadmaps studying the SoS development agenda for the coming years, like CPSoS[7] [2013-2016], Road2SoS[8] [2011-2013], CyPhERS[9] [2013-2015], T-AREA-SoS[10] [2011-2013] and several systematic literature reviews and studies

[2] H2020: Digital agenda on Systems-of-Systems, https://ec.europa.eu/digital-agenda/en/system-systems

[3] http://cordis.europa.eu/project/rcn/110028_en.html

[4] http://cordis.europa.eu/project/rcn/100874_en.html

[5] http://cordis.europa.eu/project/rcn/110001_en.html

[6] http://cordis.europa.eu/project/rcn/100096_en.html

[7] Towards a European roadmap on research and innovation in engineering and management of Cyber-Physical Systems-of-Systems

[8] Development of strategic research and engineering roadmaps in Systems of Systems Engineering and related case studies

[9] Cyber-Physical European Roadmap and Strategy

[10] Trans-Atlantic Research and Education Agenda in System of Systems

Table 1 System-of-Systems definitions

Reference	System-of-Systems definition
International Organization for Standardization [62], SoS Annex - 2015	SoS brings together a set of systems for a task that none of the systems can accomplish on its own. Each CS keeps its own management, goals, and resources while coordinating within the SoS and adapting to meet SoS goals.
International Council on Systems Engineering (INCOSE) [33] - 2015	System-of-interest whose elements are managerially and/or operationally independent systems. These interoperating and/or integrated collections of systems produce results unachievable by the individual systems alone.
Mo Jamshidi [63] - 2009	SoS are large-scale integrated systems which are heterogeneous and independently operable on their own, but are networked together for a common goal.
John Boardman et al. [17] - 2006	Conform to *system* definition, an SoS consists of parts, relationships and a whole that is greater than the sum of the parts, but with the distinction coming from the manner in which parts and relationships are gathered together and therefore in the nature of the emergent whole.
Mark Maier [75] - 1998	SoS is a class of systems which are built from components which are large scale systems in their own right.

[31][92][50][68] emphasize that the research in the SoS field is still at early stages and there is still a need of standards for SoS definition, taxonomy on SoS types and SoS architecture processes; as well as accurate methods, languages and tools to address SoS complexity specially at the software architecture level.

Enhancing Systems-of-Systems (SoS) modeling

Systems-of-Systems are usually defined as Systems composed of distributed, independent monolithic Constituent *Systems* (CSs) that interact to accomplish a mutual goal that none of the CSs can achieve individually. A *System* itself, as defined by the International Organization for Standardization -ISO 15288- in [62] is a: "Combination of interacting elements organized to achieve one or more stated purposes". Considering the absence of a precise definition for SoS, we present in table 1 several widely accepted definitions from recent literature reviews and conference papers.

Moreover, to differentiate SoS from monolithic systems, many researchers attempt to define them by their differentiating/distinguishing/specific characteristics, in particular Maier [75] identifies the following five essential characteristics, detailed in the first part of table 2 and referred to by the acronym "OMGEE": Operational independence of the CSs, Managerial

Table 2 Systems-of-Systems "differentiating characteristics" from "regular" systems

Characteristic	**Description**
Operational independence of the CSs	Each CS work independently to achieve its own individual goals and cooperatively with the other CSs to realize the SoS global goal.
Managerial independence of the CSs	CSs in an SoS belong to different organizations/companies, and each CS may be managed independently by the organization to which it belongs.
Geographic distribution	The CSs of the SoS are dispersed, they do not exchange mass or energy, they exclusively communicate information.
Emergent behavior	SoS global functionalities and purposes do not remain in any CS, they emerge from the cumulative actions and interactions between these CSs.
Evolutionary development	The SoS format is not stable, its development is subject to several evolutions (insertion, modification or suppression) of its CSs, requirements and functionalities.
Autonomy	Each CS is free to make its decisions and achieve its goals independently or together with the other CSs.
Belonging	Each CS plays a role in enhancing the value of the SoS's objective to be part of this SoS.
Connectivity	Legacy and newly added CSs interact in order to fulfill the SoS global goal(s).
Diversity	The SoS benefits from the numerous and various functions of its CSs.
Emergence	The emergence in SoS arises from the interactions between the different CSs resulting in expected or unexpected, beneficial or detrimental consequences beyond the foreseen designed behaviors.

independence of the CSs, Geographic distribution, Emergent behavior and Evolutionary development. Boardman et al. [17] define five other characteristics referred to by the acronym "ABCDE": Autonomy, Belonging, Connectivity, Diversity and Emergence as detailed in the second part table 2. Further aspects that differentiate SoS engineering from monolithic systems engineering can be found in [31][92][94].

Maier's and Boardman's characteristics overlap and denote somehow analogous properties. In this thesis, we adopt Maier's characteristics since his paper is considered as the "most recognized source on SoS engineering" [30] and the features he defined are widely recognized in the SoS world [92]. Consequently, we define and characterize an SoS by the existence of the OMGEE characteristics.

These specific characteristics introduce considerable challenges into SoS engineering. In table 3 we present seven "pain points" identified by Dahmann [29] and resulting from the INCOSE SoS working group analysis of the SoS engineering challenges. Likewise, Nilson et al. [92] reviewed several SoS engineering challenges and grouped them into five categories as described in table 4.

Table 3 Systems-of-Systems engineering "pain points"

Pain Point	**Description**
SoS thinking	Challenges in defining the key principles for SoS reasoning
SoS authority	The managerial independence of SoS poses authority and collaboration pattern challenges among the CS organizations
Leadership	Independent authorities and financial resources rise the challenge of characteristics and roles that CS leaders should possess
CS perspectives	CSs efficient integration challenges
Capabilities and requirements	Challenges in adjusting SE approaches to tackle SoS requirements and capabilities
Emergence, independence and autonomy	Challenges in adapting SE approaches to address emergent behavior and CS independences
Testing, validation and learning	Challenges in adapting SE approaches to cover SoS testing and validation

Among these challenges, we focus our research in this thesis to SoS architecture modeling and simulation challenges taking into account the general SoS research challenges such as those of defining methods, processes and tools to address SoS characteristics, interactions and emergent behavior. By tackling those challenges, we cover some of the previously described pain points such as adapting Software Engineering (SE) approaches to address SoS emergent behavior and CSs managerial and operational independence.

Many recent literature reviews [90][92][50][87], as well as the United State department of defense guidelines for SoS [95], recommend Model-Driven Engineering (MDE) approaches for SoS development as a key solution to overcome the SoS complexity and emergent behavior at the architectural level of the SoS development life-cycle.

Actually, for many decades, models helped humans to understand and analyze complex systems by reducing the level of details of the modeled system to design abstractions comprehensible by human cognitive capacities. Software engineers use models to exchange their understanding of the system's structure and behavior, as well as to predict some relevant characteristics of the modeled system [115]. Furthermore, software models play a critical role in engineering complex systems to make sure that the implementation corresponds to

Table 4 Systems-of-Systems engineering challenges

Category	Challenges
SoS research challenges	Develop methodologies to address SoS emergent behavior and evolution; introducing methods and tools to describe and analyze SoS interactions and functionalities; propose methods and tools to anticipate unintended behavior arising from SoS interactions; etc.
Modeling and architecture	Define modeling languages to conceptually describe and evaluate SoS architectures, to represent the individual independent CSs and properly define their interfaces and interactions, to describe the autonomy, distribution and interactions of CSs using agent-based approaches; etc.
Simulation	Display the characteristics of SoS while simulating its architectures, specially to identify SoS emergent behavior; show and analyze the SoS behavior, functionalities and interactions; etc.
Testing	Methodologies, techniques and tools to address complexity, standards applicability and dynamic evolution challenges when testing very large SoS; use of model-based approaches for SoS development, verification and validation; etc.
Verification	Define new and extend existing approaches and tools to verify SoS engineering; etc.

the specifications. Consequently, modeling modern complex SoS seems to be an inevitable way to manage their complexity [115].

A *software model/architecture*, as defined by ISO 42010 [60], is a representation of the structural and behavioral aspects of a software including its related artifacts such as components, relationships and environment. Models are generally described with one or more modeling languages. Model-Driven Engineering (MDE) is an effective software development approach for defining and exploiting abstract models of software systems [15][43][72], mainly due to the following fundamental advantages:

- Definition of powerful graphical modeling languages;
- Automatic generation of tools for the defined languages (e.g. graphical editors);
- Definition of platform independent models which allows a potential reuse of the models;
- Early validation of the models by transforming them into executable code (platform specific models) using Model-to-Text transformations;

- Iterative refinement of architecture alternatives enabled by the relatively smooth transition between the models' design and their execution through the (semi)automatic generation of code;

Even though the movement towards the use of model-based approaches for SoS engineering is growing, it is still at its early stages. The literature studies in [87][50][99] have reviewed several languages for SoS software architecture modeling and concluded that Domain Specific Languages (DSL) seems to be the best-accepted and mostly used approach for designing SoS architecture models. ISO 42010 [60] defines DSL as any form of expression intended to represent a software architecture within a specific domain of application. Despite the work and advances already made, newly defined and extended DSL for SoS architecture modeling still lack several concepts to cover SoS characteristics [54][49] such as: representation of managerially and operationally independent CSs and the functionalities on their interfaces, description of CSs interactions in a way to permit the analysis of the emergent behavior, modeling of SoS non-functional properties, etc.

Problem 1: Most existing Domain Specific Languages (DSL) lack concepts to model Systems-of-Systems (SoS) taking into account their specific characteristics.

Introducing security into SoS modeling

Besides the challenge of modeling SoS architectures taking into account their 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 thesis, we focus on SoS security. Indeed, as we argue in our publication [52], SoS suffers from traditional security concerns related to their complex CSs, as well as additional emergent security challenges arising from their specific characteristics [52][28][90] as synthesized in table 5.

SoS security has gained growing importance and attention over the years. Being able to successfully engineer SoS security is a prerequisite for the complex software-intensive and highly risked solutions of the modern society towards which we are moving on. After a rigorous analysis of SoS topics by European SoS experts over nineteen months, security was nominated among the twelve research themes of the SoS strategic agenda for 2020 [37]. Many other researchers have designated security as an urgent critical attribute to consistently address when engineering SoS[45][90][32][78].

Traditionally, security concerns encompass three main objectives [6]: *Confidentiality*, *Integrity* and *Availability* of the information and resources, referred to by the acronym CIA. To address them, security solutions offer security mechanisms such as *authentication* and *Access*

Table 5 System-of-Systems security engineering challenges

SoS characteristics	Specific security concerns
Managerial independence	Activities of one CS may impact the security of the other, such as access control roles and rights, trust, etc.
Operational independence	Conflicting security policies, potential incompatibilities between CSs security protocols, propagation of attacks, etc.
Geographic distribution	Different national regulations, etc.
Emergent behavior	Unintended unforeseen/emergent security issues arising from the SoS interactions and emergent functionalities, etc.
Evolutionary development	Information access, secure communication protocols and security configuration issues (new open ports) rising from the evolutionary state of the SoS (new/updated software/protocol versions).

Control, in accordance with the security objectives to overcome security *vulnerabilities, threats* and *attacks*. In table 6 we present the ISO 27000 [61] definitions of the previously mentioned security concepts.

Over the last years, there has been a growing awareness of SoS being exposed to severe security attacks. Their large scale infrastructure, complexity of their architectures and interconnections between their independent CSs makes them enormously vulnerable to malicious attacks. The number of security cyber incidents targeting SoS application domains is exceedingly increasing, 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[11] in 2016 targeting the Internet-of-Things SoS application domain; the Stuxnet attack[12] in 2010 and its successor Irongate in 2016 and the Northeast Blackout[13] in 2003 targeting the smart grid SoS; the rapid transit shutdown in San Francisco bay area[14] in 2013; the Google building attack[15] 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 (CSs) 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. In fact, many of the CSs may suffer from small, known and unresolved

[11] money.cnn.com/2016/10/22/technology/cyberattack-dyn-ddos/index.html
[12] http://large.stanford.edu/courses/2015/ph241/holloway1/
[13] https://www.scientificamerican.com/article/2003-blackout-five-years-later/
[14] http://www.bizjournals.com/sanfrancisco/blog/2013/11/bart-system-shut-down-by-software.html
[15] https://www.wired.com/2013/05/googles-control-system-hacked/

Table 6 Basic security concepts taxonomy

Security concept	**Definition**
Confidentiality	Property that information is not made available or disclosed to unauthorized individuals, entities, or processes.
Integrity	Property of accuracy and completeness.
Availability	Property of being accessible and usable upon demand by an authorized entity.
Authentication	Provision of assurance that a claimed characteristic of an entity is correct.
Access Control	Means to ensure that access to assets is authorized and restricted based on business and security requirements.
Vulnerability	Weakness of an asset that can be exploited by one or more threats.
Threat	Potential cause of an unwanted incident, which may result in harm to a system or organization.
Attack	Attempt to destroy, expose, alter, disable, steal or gain unauthorized access to or make unauthorized use of an asset.

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 but 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 [47].

In the context of this thesis, 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).*

Furthermore, it is important to mention that in our work we address the *unknown known vulnerabilities* category. Indeed, Rumsfeld [113] classify the knowledge in three categories: *known knowns, known unknowns, unknown unknowns*. Sawyer [110] added a complementary category, on which we will focus is the *unknown knowns*. He defined it as "knowledge that a person holds, but which they withhold". These categories could be projected onto vulnerability categories. In the context of SoS, each CS may define a list of its own vulnerabilities

(*known*). Since there is no central authority in the SoS, this list of vulnerabilities will remain *unknown* by other CSs, hence the nomination *unknown known vulnerabilities*.

Model Driven Engineering is a fundamental solution to address and tackle SoS security challenges via security by design, abstractions and automations [90][66][43][9]. Indeed, higher level of abstractions allow better engineering security and provide the foundations for (semi)automated analysis of the SoS security. Security should be integrated and analyzed during the design of SoS and not in the aftermath to avoid extensive functional, social and economical damages of the possible high-impact cyber attacks.

Despite its importance, security is still an immature SoS research topic [54][50]. Guidance for software engineering of SoS is relatively silent on security [37], and until now there are no standards specific for SoS security [31]. Integrating security into SoS architectures is a challenging task that should be addressed by any proposed SoS modeling method, language and tools [28][90].

Problem 2: Lack of software engineering methods, languages and tools that address Systems-of-Systems (SoS) security at the architecture phase of the SoS engineering life cycle.

Analyzing secure SoS architectures

After arguing the suitability of modeling approaches to design SoS architectures taking into account their specific characteristics and security concerns, we tackle the additional challenge of the execution of the modeled SoS architectures to analyze their behavioral aspects. For example, the emergence characteristic of the SoS could be expressed, at the architectural phase, as desirable or undesirable behaviors with the needed CS interactions to realize these behaviors. However, the behavior itself and the related unknown emergent issues become really apparent, and consequently analyzable, only through the execution of the modeled architectures [92].

Model simulation is one of the fundamental techniques to analyze and understand complex system architectures. In SoS engineering, simulation has been identified as a powerful tool to analyze SoS interactions and emergent behavior, as well as to support the early discovery and anticipation of unforeseen issues and failures [92][10][125][129]. Furthermore, the US Department of Defense guidelines for SoS engineering recommend modeling and simulation as effective approaches to address SoS complexity and emergent behavior [95].

Nonetheless, only few simulation approaches have been developed or extended to simulate SoS architectures, where the basis of the described models is the SoS behavior or the interactions among the CSs [10]. Moreover, in order to efficiently simulate and analyze SoS security emergent behavior, their specific characteristics and security-related concerns need

to be properly modeled in the SoS architectures and accurately aligned with the executable artifacts [1].

Problem 3: Need of approaches to properly align and map platform independent secure Sytems-of-Systems (SoS) architectures to executable artifacts and effectively simulate these architectures to detect and analyze emergent security behaviors such as SoS cascading attacks.

Objectives and research questions

The work of this thesis is therefore essentially focused on secure SoS architecture modeling and analysis. We aim to investigate how software engineering approaches can be extended to model and analyze secure SoS solutions for discovering emergent behaviors, precisely high-impact cascading attacks, at the architectural stage of the SoS development life cycle. Discovering such attacks at early stages, enables their analysis and resolution with smaller cost and time expenses, as well as SoS protection against cyber attacks and massive resulting damages.

This objective will be achieved by answering the following research questions (RQs) arising from the previously described problems:

- *RQ1 (to address the problems 1 and 2): How can Systems-of-Systems structure and behavior be modeled, with a particular focus on security aspects, so as to allow the prediction of unknown security emergent behaviors?*
- *RQ2 (to address problem 3): How can the modeled secure Systems-of-Systems architectures be simulated and consequently analyzed, so as to discover security cascading attacks emergent behaviors?*

Contributions

Our main contributions in this Ph.D. is a method called **S**ystem-**o**f-**S**ystem **Sec**urity (SoSSec) devised to answer our main two RQs, as shown in figure 1. The proposed method, in accordance to Ramsin's definition [109], is defined by a set of modeling conventions (the modeling language), the corresponding tools and an utilization process to guide the use of the language and tools. SoSSec is built following Model Driven Engineering guidelines and introduces several contributions regrouped into two main parts:

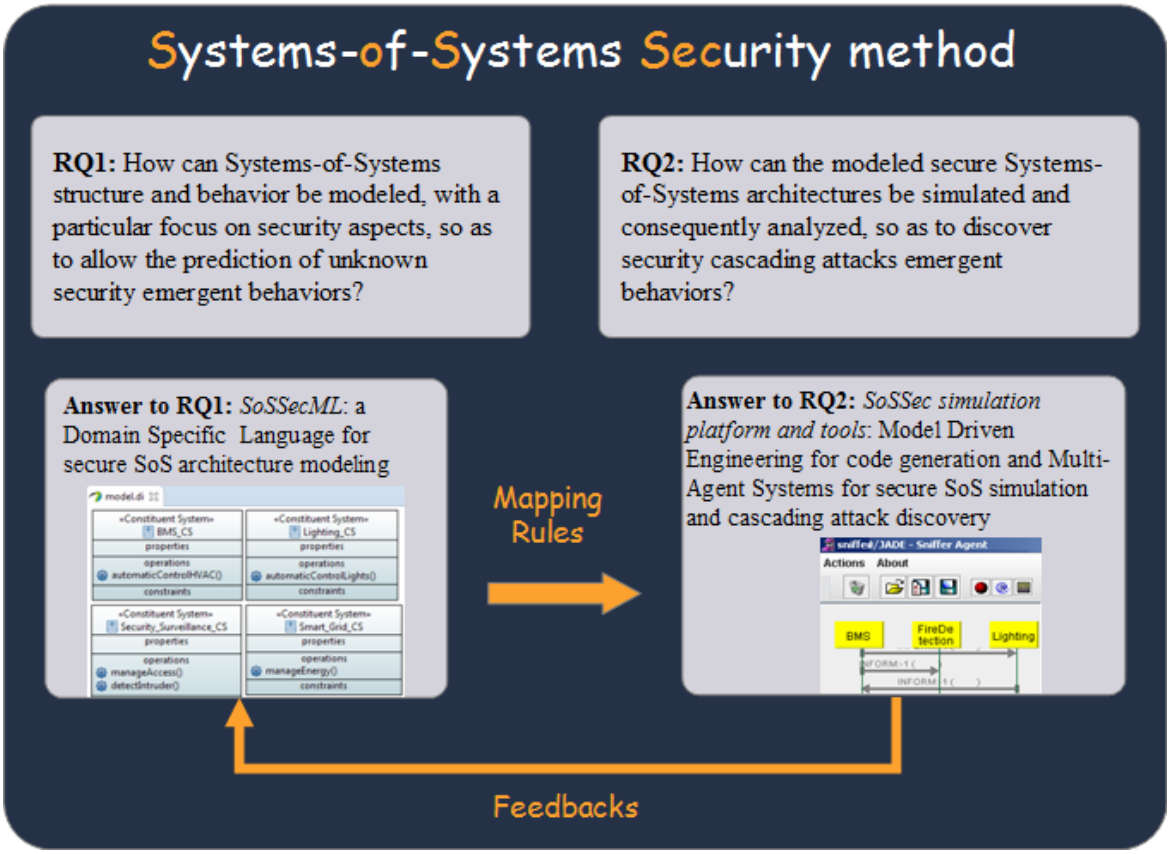


Fig. 1 SoSSec method for secure SoS modeling and analysis

1. Secure System-of-Systems architecture modeling: To answer the first research question we propose a model driven method encompassing a language along with its graphical tool, and an utilization process:

    - SoSSecML: A Domain Specific Language (DSL) that allows SoS modeling taking into account its specific characteristics and emergent security concerns. SoSSecML extends an existing DSL with concepts specialized for SoS such as constituent system and organization, and others particular for security cascading attacks such as vulnerability, pre and post conditions and security mechanism. The extensions are implemented using MDE mechanisms and tools;
    - A SoSSecML graphical editor: A visual modeling tool that extends an open source tool. We benefit from MDE transformation mechanisms to automatically generate our tool, offering by that a portable, customizable and graphical editor for secure SoS architecture modeling;
    - An utilization process to guide the use of SoSSecML and its tool: A process to provide guidance on the use of the language, along with the corresponding actors

and required tools. Therefore, we propose an utilization process for SoSSecML that describes the steps, actors and tools to be followed and used when modeling a secure SoS using our SoSSec method. We follow this utilization process to model a real-life smart building SoS: the Adelaide Health and Medical School (AHMS) smart building;

2. Secure System-of-Systems architecture analysis: To answer the second research question, we explore agent based approaches and propose an extension of a Multi-Agent Systems MetaModel and simulation platform along with the corresponding generator tool (from models to simulation artifacts) and utilization process:

   - An extension of a Multi-Agent System (MAS) MetaModel to execute secure SoS architectures in order to discover and analyze emergent unknown security cascading attacks: We argue the usefulness of MAS for SoS simulation and analysis due to the similarities between MAS and SoS concepts. Accordingly, we use the MAS MetaModel existing concepts to express the semantics of SoSSecML structural and behavioral concepts, and we extend them with security-related concepts to express SoSSecML cascading-attacks related concepts, in addition to a matching mechanism that helps identifying the possible concatenation/sequence of triggered vulnerabilities. Afterwards, we use this extended platform to simulate the secure SoS architecture models designed using SoSSecML, to analyze the CS interactions and discover the unknown possible cascading attacks resulting from those interactions;
   - A model driven based generator: We follow the MDE approach and use its mechanisms to define a set of Model-To-Text transformation rules to (semi)automatically map the secure SoS architecture to executable artifacts;
   - An utilization process to guide the use of the SoSSec generator, simulation platform and analysis mechanism: We follow this process to execute the AHMS smart building secure architectures and analyze the results to detect the related emergent cascading attacks.

These contributions are tied under the two main parts of our methods as shown in figure 1. One of the main advantages of building SoSSec following the MDE approach is offering a method with an iterative feature, guarantying an automated mapping (*Mapping Rules* in figure 1) between the modeling and analysis phases, thereby allowing the analysis of several secure SoS architectures until reaching an acceptable security level. In fact, each architectural alternative represents the software architecture of a SoS, in terms of design decisions and

the artifacts to fulfill these decisions, at a given time and for a described scenario [118]. Whenever a secure SoS architecture alternative is modeled and analyzed to discover possible cascading attacks related to this alternative, the software and security architects and security analyst decide whether the alternative is acceptable in terms of compliance to the needed security level, or there is a need to modify it and/or define some security mechanisms and go through the whole process again (*Feedbacks* in figure 1).

Thesis structure

This thesis manuscript is composed of two main parts. The first part represents the related work, contributions and case study application for the SoSSec modeling phase. The second part represents the related work, contributions and case study application for the SoSSec analysis phase.

Structure of the Part I - Secure SoS architecture modeling

The first chapter of this part presents our related work study of the existing approaches that address the *modeling* of *secure Systems-of-Systems* architectures, followed by a deeper investigation of the related work targeting the SoS modeling and/or security. The results sustain the need for a new modeling language extension to model SoS taking into account their specific properties. They also highlight the fact that the extension should introduce security concepts to allow the security modeling and analysis of SoS architectures, specially that of the cascading attack emergent behavior.

In the second chapter, we propose the SoSSecML, a DSL for secure SoS architecture modeling, to fill the gaps found in the related work. The chapter details the Domain Specific Language (DSL) of our Systems-of-Systems Security Modeling (SoSSec) method, and its corresponding graphical editor tool, as well as the tool construction process to guide the construction/modification of the tool, and an utilization process to guide the use of SoSSec and its first tool (the graphical editor), answering by that our first research question RQ1.

In the third chapter, we exemplify the modeling phase of the SoSSec method (language, tool and utilization process) by applying it on a case Study, the Adelaide Health and Medical School (AHMS) smart building. It presents the details and results of using the proposed SoSSecML along with its graphical editor to model the secure software architecture of the AHMS.

After the secure SoS architecture modeling part, the second part includes the analysis phase of our SoSSec method. This part presents the execution/simulation of the modeled architectures to analyze their behavior, in particular the unknown security cascading attacks.

Structure of the Part II - Secure SoS architecture analysis

The first chapter of this part presents our related work study of the existing approaches that jointly address the *modeling* and *analysis* of *secure Systems-of-Systems* architectures. It is supplemented by a deeper study of the related work targeting the SoS analysis. This study shows the limitations of the existing approaches. Moreover, the study shows that a Multi-Agent System (MAS) simulation method is a solid base for SoS architecture analysis. It also highlights that the use of MAS in our method should introduce well-defined relations between the secure SoS models developed using our SoSSecML language, and their realizations/execution. In addition, the related work study shows the need of extending MAS to include security executable artifacts, in order to express the SoSSecML security concepts.

The second chapter presents our contributions to answer the limitations of the related work in analyzing secure SoS architectures. The chapter presents the analysis phase of our SoSSec method. It describes an extension of a Multi-Agent System (MAS) MetaModel, a set of Model-To-Text (MTT) transformation rules - the SoSSec generator tool - and a vulnerability matching mechanism, to analyze a secure SoS architecture and predict the possible sequence of triggered vulnerabilities forming high-impact cascading attacks.

The third chapter exemplifies the SoSSec analysis part by using it to analyze the secure architecture of the AHMS smart building SoS. It describes the use of SoSSecML along with the generator tool and the extended MAS simulation platform, according to its utilization process, to analyze the secure software architecture of the AHMS and predict the related possible cascading attacks. We concludes this chapter by a discussion of our AHMS case study limitations.

Part I

Secure System-of-Systems (SoS) Architecture Modeling

Part I Introduction

Systems-of-Systems Engineering (SoSE) has drawn increasing attention since SoS application domains and security concerns became associated to many of our modern society solutions. A large number of SoSE challenges and open issues, related to the different phases of SoS development life cycle, were pointed out by recent literature reviews on SoS, as noticed in the general introduction. These challenges call for special attention when engineering SoS such as addressing non-functional properties, in particular security; bridging different software engineering solutions to engineer secure SoS and offering tool support.

In this thesis we focus on the SoS modeling and analysis challenges taking into consideration security concerns. At the moment of writing this thesis, only a very limited number of approaches address secure SoS architecture modeling and analysis. Moreover, these recent approaches are still at the very early stages and do not present concrete solutions.

In this first part, we address the secure SoS modeling challenge. We propose a **S**ystems-**o**f-**S**ystems **Sec**urity **M**odeling **L**anguage called SoSSecML along with its graphical editor and utilization process, to answer our first research question (RQ1). The contributions are organized as follows: In chapter I.1, we review existing approaches for secure System-of-Systems (SoS) architecture modeling and we identify their limitations. To overcome these limitations, we investigate the Model Driven Engineering approach and propose SoSSecML along with its graphical editor and an utilization process of the language and tool in chapter I.2. In chapter I.3, we followed the guidance of the utilization process to illustrate our contributions by modeling the secure architecture of a real-life smart building SoS, the Adelaide Health and Medical School (AHMS) building. We conclude this part by a discussion of the advantages of our SoSSec method.

Objective: Propose a model-driven method with a modeling language, tools and utilization process, to model Systems-of-Systems structure and behavior with a particular focus on security aspects that allow the prediction of unknown security emergent behaviors such as cascading attacks.

Chapter I.1

Related Work: Secure SoS Architecture Modeling

I.1.1 Introduction

To gain insight into the current status of secure SoS modeling research, we conducted a review of the related existing approaches that have been published up to this date[1].

Since our objective in this first part is to **model** the SoS architectures taking into account security concerns to answer our first research question RQ1, we report in this chapter the existing approaches that jointly or partially address (*SoS* and/or *security*) and *modeling* (categories A, B and C in table I.1.1);

It is important to mention that the publications presented and analyzed in this related work chapter were extracted from several bibliographic databases, mainly: Scopus[2], IEEE Xplore[3], ACM digital library[4], Springer[5] and Wiley online library[6]. The extraction of the publications was made in a structured way by using appropriate keywords related to our topics of interest as described below, in addition to several inclusion and exclusion criteria to select relevant studies, however this study is not a complete Systematic Literature Review [67]. Moreover, this related work was analyzed having in mind our research questions that we formulated in the general introduction and the sought objectives of this thesis, instead of only the limitations identified by their own authors.

[1] July 2017
[2] https://www.scopus.com/home.uri
[3] Institute of Electrical and Electronics Engineers Xplore, http://ieeexplore.ieee.org/Xplore/home.jsp
[4] Association for Computing Machinery, http://dl.acm.org/
[5] https://link.springer.com/
[6] http://onlinelibrary.wiley.com/

In table I.1.1 we present the reviewed approaches and analyze them regarding the following topics of interest:

- **SoS**: This thesis' focal point is Systems-of-Systems, thus the reviewed studies address, at least partially, this topic. For the data extraction of our review, we used the following keywords to cover this topic: "System-of-Systems" or "Systems-of-Systems" or "System of Systems" or "Systems of Systems". Moreover, some of the related work does not mainly focus on the SoS specific characteristics, but rather consider "large-scale" or "cyber-physical" systems as SoS due to their complexity. Some other studies focus on one specific SoS "application domain". We included the latter type of related work when we found it interesting regarding the three other topics (modeling, security and analysis) and possibly applicable/extensible to SoS;

- **Security**: Among the challenges related to SoS non-functional properties, we focus in this study on security due to its importance in the SoS context. Furthermore, we give a specific attention to concepts related to cascading attacks related concepts by virtue of its particular specificities, related to the CS interactions and emergent behavior characteristic of the SoS, and its high impact in such complex context. Accordingly, we considered the "security" and "attack" keywords in our search;

- **Modeling**: As argued in the general introduction chapter, Model Driven Engineering (MDE) is an effective approach to deal with the SoS complexity as well as security related modeling and evaluation challenges. In this thesis, we are firstly interested in proposing a method and tools to conceive secure SoS architectures. Consequently, to include this topic in our review, we used the following keywords: "architecture" or "design" or "model driven engineering" or "model driven architecture";

We organized the studied approaches in three categories:

1. Category A: The modeling approaches that cover at the same time SoS and security modeling (SoS AND Security AND Modeling);

2. Category B: The modeling approaches that only cover SoS architecture modeling (SoS AND Modeling);

3. Category C: The modeling approaches that address secure "regular" system modeling (Security AND Modeling);

As shown by our review of related work (table I.1.1), at the beginning of this thesis there was only one proposal (presented in two papers) covering the *SoS*, *modeling* and *security*

Table I.1.1 Secure Systems-of-Systems **modeling** approaches

Category	Studied approach	SoS	Security
A	Merabti et al., 2015 [79]	✓	✓
	Merabti et al., 2011 [78]	✓	✓
B	Zhang, 2015 [130]	✓	✗
	Cavalcante et al., 2013 [22]	✓	✗
	Oquendo et al., 2016 [99]	✓	✗
	Bryans et al., 2014 [21]	✓	✗
	Faldik et al., 2017 [41]	✓	✗
C	Jurjens et al., 2002 [64]	✗	✓
	Lodderstedt et al., 2002 [73]	✗	✓
	Elahi et al., 2007 [39]	✗	✓
	Munante et al., 2013 [36]	✗	✓
	Apvrille et al., 2016 [5]	✗	✓

topics, that is why we further explored the *SoS* **modeling** related work. We present in section I.1.2 a review and analysis of the approaches that jointly address *SoS modeling* and *security* (category A in table I.1.1). This is followed by a deeper analysis and classification of *SoS modeling* related work (category B in table I.1.1) in section I.1.3, and a review and analysis of the existing work on system *security modeling* (category C in table I.1.1) in section I.1.4. The literature related to the research findings is further discussed and concluded in section I.1.5.

I.1.2 Existing approaches for secure SoS architecture modeling

In recent years, diverse studies in the field of SoS have been published, many of them highlighting the urgency/importance of considering security when engineering SoS [90] [45] [38]. In spite of this, up to this date and to the best of our knowledge, SoS security is barely addressed at the architectural phase of the development life cycle. As shown in table I.1.1, the review that we conducted resulted in only one approach (category A) that discusses the *modeling* of *secure SoS* architectures (in addition to those discussed previously).

Merabti et al., in their study of the 21st century challenges for critical infrastructure protection [78], consider the importance of security design when architecting critical infrastructures such as smart grids. The authors detail the challenges of protecting such infrastructures, from complexity and constraint environment, to security and crisis management. In addition to suggesting solutions to address these challenges, this study asserts the

necessity and usefulness of considering vulnerabilities and security issues when designing SoS solutions. However, the authors do not clearly detail in a concrete way, what or how software engineering methods and tools could be used, extended or defined to handle the suggested thoughts.

In a more recent work [79], Merabti et al. propose a cryptographic key management schema that considers smart grid security requirements. However, the solution is restrained to secure the data exchanges between two sub-systems (the smart home area network and the neighborhood area network) addressing in this way a single security concern without considering the SoS characteristics.

Given the lack of existing approaches that jointly address *SoS*, *security* and *modeling* topics, we extend our review to study and analyze the state of the art of *SoS modeling* (category B in table I.1.1) and systems *security modeling* (category C in table I.1.1), respectively in the following two sections.

I.1.3 Existing approaches for SoS architecture modeling

Traditionally, many approaches were used to model the software architectures of a system, such as: Formal approaches that describe the software architecture models using mathematical expressions and predicates; goal oriented approaches which focus on goals to capture the functional and non functional system requirements in the software architecture models; and Architectural Description Languages (ADLs) which are languages used to describe system software architecture, like its components, connectors, rules and guidelines. Some of these system architecture modeling approaches were used or extended to represent SoS architectures.

We selected some of the approaches attempting to model SoS in each of the previously mentioned categories, and we analyzed their suitability for modeling SoS specific characteristics as recommended by Maier [75].

Some formal methods (e.g. graph based-methods: Bayesian Networks, Functional Dependency Network Analysis seem well suited to model the geographic distribution and interactions between CS but they are still limited compared to ADLs regarding the representation of the structural architecture (like components; interfaces; contracts and ability to describe hierarchical levels of details).

Goal oriented approaches such as Keep All Objectives Satisfied and agent-based approaches could be used to model SoS global goal and CSs individual goals, as well as behavioral aspects like the the interaction between different CSs. However, like in the case of formal methods, goal oriented approaches are not the best suited for describing SoS

structural architecture. Besides, seeing that another crucial aspect in the context of our work is the ability to model security mechanisms, properties and concepts like vulnerabilities and attacks, ADLs security extensions presents more useful modeling concepts than goal oriented approaches.

For these reasons, ADLs seems to be a well suited basis for the extension/definition of a domain specific modeling language for secure SoS architectures modeling (this analysis and conclusion are published in [54]).

Other recent systematic literature reviews [50][87] studying the usefulness of existing modeling approaches for SoS modeling also concluded that ADLs seems to be the best-accepted and mostly used approach for designing SoS software architecture models.

Given the diversity in the recently proposed approaches for SoS modeling and on the basis our study and recent reviews outcomes, we restrain our review to ADLs, we present and analyze below the latest ADLs intended to model SoS software architecture, and those issued from recent International and European SoS projects like AMADEOS[7] (2013-2016), DANSE[8] (2011-2015) and COMPASS[9] (2011-2014).

Zhang in [130] introduce an approach based on several steps, going from SoS architecture (and more specifically Cyber-Physical Systems) specification to its simulation. The author then only details the specification part, where Cyber-Physical Systems are designed through a combination of the Architecture Analysis and Design Language (AADL), the modelica language, and formal specifications. However, this approach is still immature and does not concretely cover the SoS characteristics. Except from the general specification step, the suggested steps are not detailed and there are no suggestions for any concrete methods and tools to realize them.

Cavalcante et al. in [22] extend the Acme ADL to model cloud application architectures. Among the introduced concepts they suggest relationships between Cyber-Physical Systems elements, contracts, etc. The proposed ADL remains a simple first-proposal language (as stated by its authors), limited for cloud SoS service providers and applications.

These two approaches are specific to particular SoS domains and remain insufficient to model generic SoS architectures taking into account their specific characteristics.

Oquendo et al. [99] propose a formal ADL (SosADL) along with a model driven tool-set to describe SoS software architectures. SosADL introduces four concepts to specify an SoS: *system*-represented by interaction points and internal behaviors; *mediator*-defined as internal and external duties, and controlled by the SoS to manage the interactions between CSs; *coalition*-temporary combination of actions among the connected systems; *environment*-

[7] Architecture for Multi-criticality Agile Dependable Evolutionary Open System-of-Systems
[8] Design for Adaptability and Evolution in Systems-of-systems Engineering
[9] COMPrehensive modeling for Advanced Systems-of-Systems

conditions related to the surroundings of the system. This newly developed approach defines expressive SoS concepts, although it remains powerless in expressing the unknown emergent behavior. The authors intend to use DEVS simulation to further explore their proposition.

Another ADL that has been predominantly used and extended to model SoS architectures is the System Modeling Language (SysML) [44]. Indeed, recent studies [50][68] that review a collection of ADLs and compare them regarding their effectiveness in modeling SoS architectures, conclude that SysML is a key used approach and seems to be suited as a basis for SoS architecture modeling, mainly due to its ability to model CS structure and interactions in terms of interfaces and constraints.

Moreover, SysML is actively used in recent SoS European projects. In the DANSE project SysML has been used to model and execute SoS architectures in order to test the obtained execution results as regards to interface contracts. In the COMPASS project, SoS Constituent Systems (CSs), their interfaces and contracts that oversee operations between them were modeled using SysML. In addition, a complementary formal language, the Compass Modeling Language (CML), was proposed to verify and analyze the contract at the interfaces [21]. In an alternative work recently published [41], the authors make use of the Object Constraint Language to improve the description and analysis of the contracts.

In both projects, the proposed approaches do not extend SysML. Nevertheless, SysML is limited regarding some of the SoS features like: the representation of CS goals and evolution, an alternative to deal with the independence of the CSs, architectural elements that represent the interactions on CS interfaces in a way to detect SoS emergent behaviors resulting these interactions [54][50].

To sum up, ADLs seem to be well suited to model SoS software architecture, but studied ADLs are still immature and incomplete when addressing the SoS specific characteristics, in particular the unknown emergent behaviors. Furthermore, our study shows that SysML is actively used and forms a solid basis for any language definition/extension addressing SoS architecture modeling. However, SysML lacks concepts to cover SoS specific characteristics such as goals, independence and emergent behavior as well as security concerns.

I.1.4 Existing approaches for secure system architecture modeling

As discussed in section 1, our review confirmed the absence of modeling languages developed to model *SoS* security at the start of this thesis[10]. However, many modeling language extensions have been developed to address *system* security modeling.

Table I.1.2 Modeling language comparison for system security modeling

	UMLSec [64]	**Secure UML [73]**	**Secure i* [39]**	**MODELO [36]**	**SysML-Sec [5]**
Security concerns	Integrity Authenticity Confidentiality Access Control	Access Control	Access Control	Access Control	Integrity Authenticity Confidentiality Autonomy
Threat-Vulnerability concepts	✗	✗	✓	✗	✗
Attack concept	✗	✗	✗	✗	✓
Goal concept	✗	✓	✓	✓(task)	✗
Context concept	✗	✗	✗	✓	✗

In this section, we review the most important extensions and analyze their modeling ability regarding criteria that we consider important in the context of secure SoS modeling, mainly to subsequently allow the discovery and analysis of cascading attack emergent behaviors. Following are the analysis criteria:

- Security concern: SoS are exposed to different security issues targeting their integrity, authenticity, confidentiality, etc. Therefore, to remain general in describing secure SoS, it is preferable to not focus on a specific security concern;
- Threat, vulnerability, attack: These concepts are important to model and analyze the SoS cascading attack security issue;
- Goal: SoS are characterized by their global goal(s), refined into individual goals, to be achieved by the interactions of CSs. To strengthen the security of SoS, a security goal should be defined and modeled to guide the analysis of security risks and the recommendation of relevant security mechanisms;

[10] October 2014

- Context: In SoS, the context/environment in which the CSs are working and interacting is a meaningful concept due to its impact on the functional as well as non-functional aspects of the SoS. Even if the context in SoS is not fully known at design-time, it could be helpful to represent it for a better understanding of its impact, specially in term of security risks and possible emergent behaviors.

In [64] , Jurjens et al. propose UMLSec, an UML based extension to express security requirements like data secrecy and integrity, information flows restrictions and Role Based Access Control.

Lodderstedt et al. [73] introduced SecureUML, another UML extension founded on Role Based Access Control, with additional support to specify authorization constraints. This approach aimed at increasing the productivity and quality of secure distributed systems.

Elahi et al. present in [39] secure i*, a goal oriented approach intended to model and analyze security trade-offs among competing goals of multiple actor systems. The authors attempt to quantitatively assess security mechanisms' impact on goals and threats.

MoDELO, introduced by Munante et al. in [36], is a MOdel-Driven sEcurity poLicy approach that extends UMLSec with the Organizational Based Access Control elements, like role, subject, object, action and context, to model and evaluate access control requirements.

In [5] Apvrille et al. propose SysML-Sec, a SysML based model driven approach for security. It is a formal approach inspired from the goal oriented KAOS (Keep All Objectives Satisfied) for requirement analysis. The SysML-Sec process is defined over three steps: System analysis (system assessment to identify security requirements and threats), system design (implementation of software security mechanisms) and system validation (formal verification and test of the designed models).

As presented in table I.1.2, the analysis of these approaches shows that none of the studied modeling language extensions for system security modeling satisfy all the investigated criteria. Hence concepts from each approach should be considered by any forthcoming modeling language aiming for secure SoS architectures modeling, such as those highlighted in table I.1.2: Security concerns (confidentiality, integrity, authenticity, access control, autonomy); threat, vulnerability and attack concepts; goal concept and context. Furthermore, the vulnerability concept should be refined and additional related concepts should be introduced to explicitly model CS vulnerabilities and their details in a way so as to allow the discovery of their cascade/sequence to predict and trace possible SoS cascading attacks.

I.1.5 Conclusion

Our review shows that none of the studied works fully addresses our main objective, that of *modeling secure SoS* architectures in a way that allows the discovery of the security cascading attacks emergent behavior, at the architectural stage of the SoS development life cycle.

Our study of existing approaches used to *model SoS* software architectures and/or system *security* shows that Architecture Description Languages (ADLs) are well suited approaches to model SoS, but newly defined ADLs for SoS are still immature and incomplete when addressing unknown emergent behaviors. Over and above, security is not payed necessary attention when modeling SoS; up to this date and to the best of our knowledge, there is no modeling language security extension developed for secure SoS architectures modeling. Moreover, the study reveals that the System Modeling Language (SysML) seems to be the most suited and actively used modeling language to design SoS structural architecture, specifically CSs and their interfaces. However, SysML lacks concepts to represent CSs managerial and operational independence, CSs interactions and emergent behavior and security, especially concepts related to vulnerabilities and cascading attacks.

All these reasons sustain the need for a new modeling language extension, reasonably based on SysML, to model SoS taking into account their specific properties. The extension should introduce also security concepts to allow the security modeling and analysis of SoS architectures, specially that of the cascading attack emergent behavior.

In the next chapter, we introduce our Domain Specific Language, called **SoSSecML**: Systems-of-Systems Security Modeling Language. SoSSecML is an ADL extension, precisely a SysML profile, proposed to answer our first research question, RQ1: *"How Model Driven Engineering approaches could be extended and used to model Systems-of-Systems structure and behavior with a particular focus on security aspects to allow the prediction of unknown security emergent behaviors?"*

Chapter I.2

SoSSec: A Model driven method for secure SoS architecture modeling

I.2.1 Introduction

As argued in the general introduction, among systems and software engineering techniques that have been extended and adapted to conceive Systems-of-Systems (SoS), Model-Driven Engineering (MDE) has gained a noticeable interest with considerable evidence of activity in both SoS research and practice area. MDE approaches are key solutions for SoS engineering to overcome the issues introduced by their complexity, emergent behavior, and non-functional properties at the architectural level of the SoS development life-cycle.

In this chapter, we investigate MDE to propose a Domain Specific Language (DSL), called Systems-of-Systems Security Modeling Language (SoSSecML), and its corresponding graphical editor tool and utilization process. SoSSecML aims to model the software architecture of SoS with a specific focus on security aspects that allow the prediction of unknown security emergent behaviors, answering by that our first research question RQ1.

SoSSecML is a DSL that extends the System Modeling Language (SysML) with concepts specific to SoS and their security. However, we make our language relevant to any SoS type, SoS application domain and security vulnerabilities. On one hand, we followed the MDE approach and the profile mechanism described in section I.2.2 to develop our visual modeling language abstract and concrete syntaxes, as respectively detailed in sections I.2.3 and I.2.4. On the other hand, we take advantage of the automatic tool generation mechanisms of MDE to produce a graphical editor that supports our language in section I.2.5. In section I.2.6 we present the utilization process that we define to guide the use of our SoSSec method and in section I.2.7 we conclude this chapter.

I.2.2 Model Driven Engineering (MDE)

Model Driven Engineering (MDE) is a software development approach that describe complex systems at multiple levels of abstraction using models. *Model* is the primary concept that guides the whole MDE development process [72]. It is a simplified *representation of* a real-world *system*, and it is conceived with a given purpose in mind. The model itself is *conform to* a more abstract level, the *MetaModel*, that defines the concepts used to describe the model and its relationships [15]. This MetaModel is also *conform to* an extra abstraction level, the *MetaMetaModel* which is conform to itself as shown in figure I.2.1.

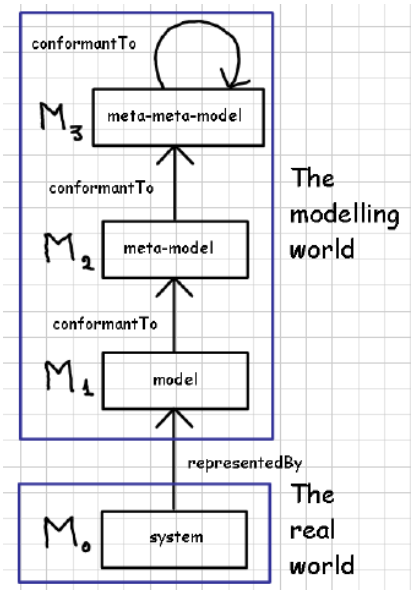


Fig. I.2.1 Model driven architecture levels from [15]

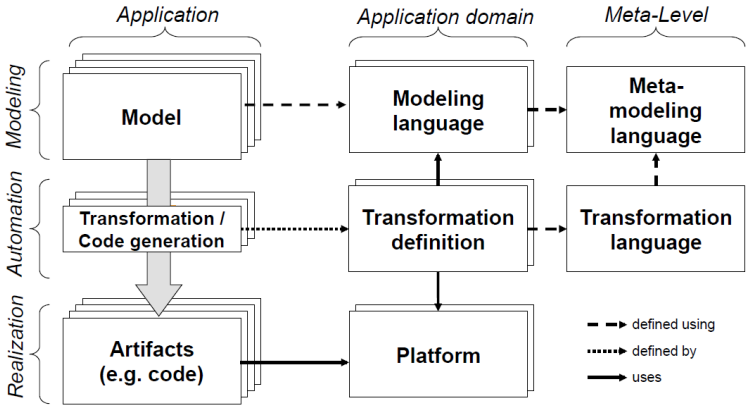


Fig. I.2.2 MDE approach from [19]

In table I.2.1, we recapitulate the definitions of the basic MDE concepts as defined by Kurtez and al. in [72].

The main principle of MDE is to create abstract models and methodically transform them into concrete implementations through automatic model transformations [43]. An MDE approach could be used to create a model (creation), check it for completeness and inconsistency (analysis), transform it into other models or platforms (transformation), test the model, execute it (simulation); or to transform a particular product to a completed model (reverse engineering).

As captured in figure I.2.2, following the MDE approach, a *model* is *defined using* a *modeling language* which is *defined by* a *meta-modeling language*, which can be defined using itself. The resulting model could be mapped to *code/artifacts* of a specific *platform* through

Table I.2.1 Definitions of MDE fundamental concepts from [72]

MDE concept	**Definition**
System	a delimited part of the world considered as a set of elements in interaction
Substitutability principle	a model M is said to be a representation of a system S for a given set of questions Q if, for each question of this set Q, the model M will provide exactly the same answer that the system S would have provided in answering the same question.
Model	a representation of a given system satisfying the substitutability principle
MetaModel	a model such that its reference model is a MetaMetaModel
MetaMetaModel	a model that is its own reference model (it conforms to itself)

transformation/code generation. These transformations are *defined by* a *transformation definition* that is *defined using* a *transformation language*.

For the reasons presented in table I.2.2 and more, MDE is increasingly related to Domain Specific Language (DSL) engineering. Indeed, a DSL is a language that defines appropriate notations and abstractions to effectively capture a problem in the specific domain it describes [34], which is in our case the secure SoS modeling and analysis. By allowing the specification, visualization, verification of a system, with a mapping between the problem world and the solution world, DSLs improve the quality, reliability, maintainability, re-usability and flexibility [72]. Moreover, the use of a DSL to model non-functional properties, like security, leads to better solutions [74][90][108][42][28].

Table I.2.2 Model Driven Engineering advantages [19] [115][15][43][72]

MDE advantage	**Description**
Support the separation of design concerns	By specifying several viewpoints and their consistency at the MetaModel level
Ensure a better understanding of the models behavior	By systematically executing the models to explore their behavior
Reduce the software developers effort	By automatically mapping the abstract models to the complex implementation platforms
Decrease complex software handcrafting accidental issues	By directly evolving models into full-fledged implementations
Support for models manipulation	By providing technologies such as model editors, transformation tools and analysis techniques
Guarantee the interoperability and portability of the models	By ensuring model interchange under the XML Metadata Interchange (XMI) format for example

We followed MDE to define our graphical DSL, SoSSecML, along with its visual modeling tool to design general secure SoS architectures, as detailed in the following sections. Having these models, we subsequently use MDE to generate the corresponding code to a specific platform and execute it to analyze the behavior of the models, in particular, the emergent cascading attack security problem as detailed in the second chapter of part II.

I.2.2.1 MDE Meta-Modeling approach for Domain Specific Language (DSL) definition

Generally, a modeling language is specified by a syntactic notation (syntax) which is a set of legal elements, together with the meaning of those elements expressed by relating the syntax to a semantic domain.

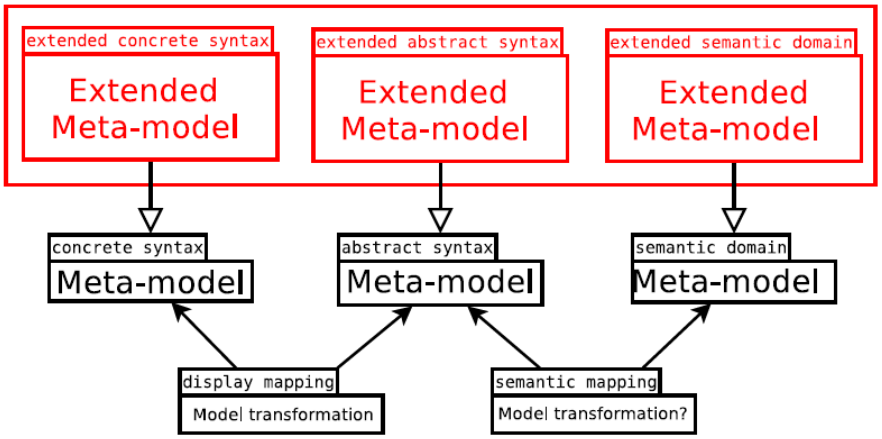


Fig. I.2.3 MetaModeling approach for DSL definition from [27]

MDE offers a MetaModeling approach to define a graphical language. The approach consists of a set of five components [72] as shown in figure I.2.3 and described below:

- Abstract syntax: Defines a set of notations (concepts and relationships) specific to the domain of the language. In MDE, the abstract syntax is represented using a MetaModel;
- Concrete syntax: Defines a set of symbols, that are used to represent the concepts of the MetaModel. A modeling language may have one or many concrete syntaxes each one denoted by a display surface MetaModel;
- Semantics: Defines the meaning of the concepts and relationships in the language. In MDE, semantics could be defined through the semantic mapping towards accurate

semantics of an existing programming language or platform. Consequently, well-defined semantics guarantee the unambiguity when executing the models;

- Display mappings: Represents the link between the abstract syntax and the concrete syntax. It is defined using model transformations;
- Semantic mappings: Represents the link between the abstract syntax and the semantics. It is defined using model transformations, mainly the code generation.

Our SoSSecML encompass these five components as it is build according to the MDE MetaModeling approach. In section 2 and 3 we respectively describe SoSSecML abstract and concrete syntaxes and their display mapping (SoSSecML modeling tool) in section 4, whereas SoSSecML semantics and the semantic mapping will be detailed in Part II. The last section of this chapter introduces the SoSSec utilization process.

I.2.2.2 Profile extension mechanism

There are three primary methods to define a DSL [115]:

1. Refine an existing modeling language: Specialize some of its general concepts to represent the domain specific concepts;
2. Extend an existing modeling language: Enrich the existing language by adding new domain specific concepts build up the existing concepts;
3. Define the new DSL from scratch.

Defining a new language from scratch encounters serious drawbacks such as the absence of support structure, difficulties related to the development of accurate and inexpensive tool support due to the sophisticated semantics of the modeling language and their interpretation. Therefore, refining or extending an existing modeling language benefits from the reuse, to a certain level, of the existing language structure and tools [115]. In figure I.2.3 the general picture of the MetaModel extension approach, consisting in extending the abstract, concrete and semantics of the reference MetaModel, is represented with red rectangles.

In the context of our work, as concluded in the related work chapter, the System Modeling Language (SysML) serves as a solid base for our DSL. We will extend SysML with new concepts specific to the Systems-of-Systems and security domains. Thus we are extending SysML to cover secure SoS modeling.

Practically, an important light-weight extension mechanism is the *profile*. A profile defines extensions to a reference MetaModel with specific domain concepts that adapt

it to this particular domain [98]. Albeit profiles are usually associated with UML, they can be generalized to other languages as well. The basic mechanisms to define a profile are *stereotypes* and *constraints*, as shown in figure I.2.4. A *stereotype* is a generalization of a *GeneralizableElement* or a specialization of another stereotype. It defines how a *ModelElement*, a metaclass of the reference MetaModel, will be extended. Stereotypes are usually supplemented by *constraints* to capture the domain specific constraints.

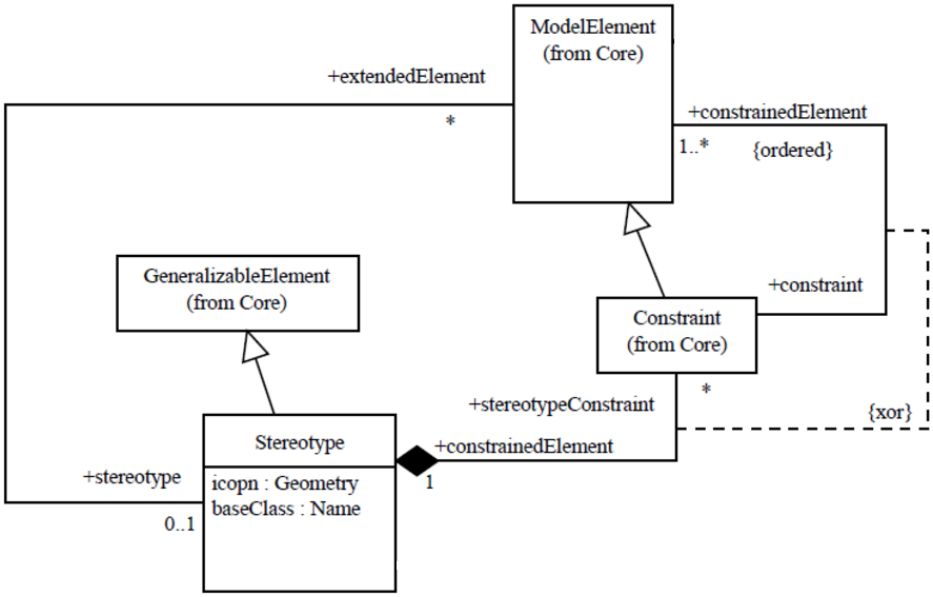


Fig. I.2.4 Profile definition mechanisms from [98]

SysML itself is a profile defined on the general purpose Unified Modeling Language (UML), to specify it to the *system* domain. For our purpose, we used stereotypes to extend SysML's MetaModel with our new concepts and the corresponding relationships between these concepts and existing SysML's concepts, as detailed in section 2.

I.2.2.3 Automatic generation of DSL tools

Each DSL needs its customized set of tools, essentially a graphical editor, a code generator and an analyzer [43]. One of the main advantages of the MetaModeling approach is the definition of these profile tools as extensions of existing tools for the base language using MetaTools. A MetaTool is a tool that supports the specification and generation of another tool, reducing by that the time, cost and effort of building new tools from scratch. Even more, the MetaTools allow the re-generation of the profile tools, whenever the DSL evolve, to incorporate the new features in an effortless, non-expensive way.

MDE MetaModeling approach offers the display and semantic mapping mechanisms to automatically generate tools as defined in the previous section. For our SoSSecML profile, we provide a graphical editor essential for RQ1 software tools, as detailed in section 4. Moreover, we propose a code generator tool towards an analysis platform, necessary for RQ2 software tools, as detailed in Part II, chapter 2.

I.2.3 SoSSec Modeling Language (SoSSecML) abstract syntax as SysML extension

As argued in the state of the art chapter, the System Modeling Language (SysML) is actively adopted and forms a good basis for SoS modeling, mainly due to its ability to represent the structure of the constituent systems. However, SysML lacks concepts to represent SoS important properties such as CSs managerial and operational independence and CSs interactions leading to emergent behavior. Moreover, SysML does not encompass security related concepts.

Therefore, to counter SysML limitations in modeling the specific SoS domain and its particular cascading attack security concern, we propose a DSL called Systems-of-Systems Security Modeling Language (SoSSecML). SoSSecML is a SysML extension defined following the MDE MetaModeling approach and using the profile mechanism. Accordingly, SoSSecML abstract syntax (MetaModel) extends the SysML abstract syntax (MetaModel) with several stereotypes.

We first give an overview of the SysML MetaModel in the following sub-section, and afterwards we detail the SoSSecML profile in the next subsection.

I.2.3.1 SysML MetaModel

SysML[1] is a graphical modeling language defined by the Object Management Group (OMG) as a profile of the Unified Modeling Language[2] (UML). The profile extends UML, mainly with the following concepts: textual requirements, block construct, activity diagrams that support behavior, constraint block, ports and information flows.

SysML is composed of nine diagrams, grouped under three categories, as presented in the Sysml specification document [97] and shown in figure I.2.5:

[1] http://www.omgsysml.org/
[2] http://www.omg.org/UML/

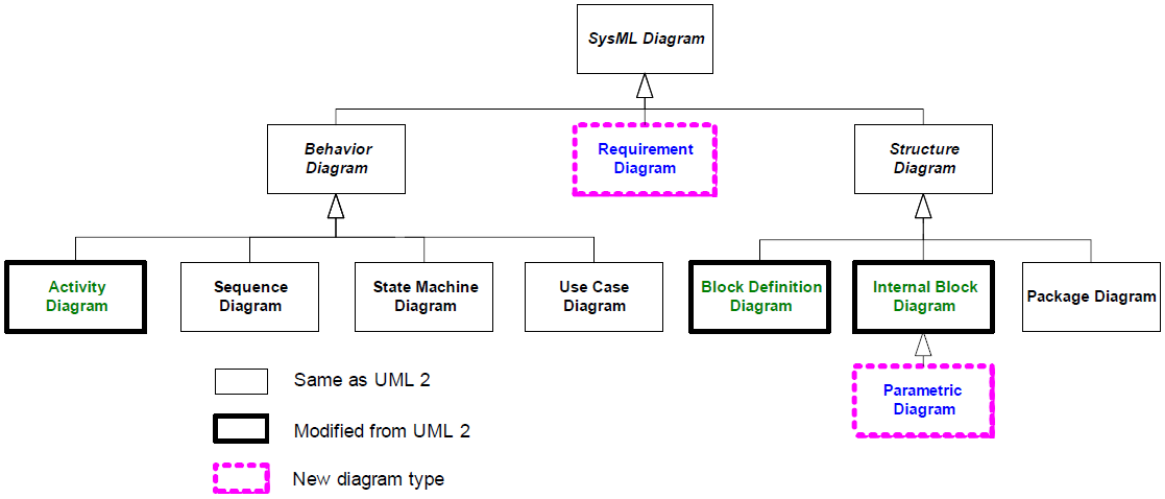


Fig. I.2.5 SysML Diagrams from [97]

- Structural diagram: To represent the structure and components of the system. It consists of the package diagram, block definition diagram, internal block diagram, and parametric diagram;
- Behavior diagram: To cover the behavioral aspects of the system. It includes activity, sequence, state machine and use case diagrams;

In our work, among the SoS engineering challenge identified in the general introduction, the first challenge that we are addressing is the modeling and architecture one, which consists in defining a modeling language to conceptually describe SoS architectures, to represent the individual independent CSs and properly define their interfaces and interactions, all that while taking into account the SoS specific characteristics and security concerns, in particular the cascading attack emergent security problem.

Accordingly, SoSSecML extends mainly two SysML diagrams: The block definition diagram (structure) and the activity diagram (behavior) to answer our first research question RQ1 consisting in extending MDE approaches to model the SoS structure and behavior while considering the security aspects that allow the prediction of unknown security emergent behaviors.

Actually, in regard to the SysML structural diagrams, the Block Definition Diagram defines the main features of blocks/system components. It captures the definition of blocks in terms of properties and operations, and relationships such as a system hierarchy or a system classification tree. The Internal Block Diagram in SysML captures the detailed internal structure of a block/system component in terms of properties and connectors between properties. The parametric diagram is a restricted form of internal block diagram that shows

only the use of constraint blocks along with the properties they constrain within a context. Therefore, we chose to extend the Block definition diagram because it allows the modeling of the SysML main modeling elements such as the hierarchy of the system (possible extension to represent the SoS); the components of the system (possible extension to represent the CS as black boxes), their interfaces and operations (possible extension to describe the CS interfaces and global functionalities on the interfaces) as well as the connectors between the components (possible extension to describe the CS interactions).

Moreover, as regards the SysML behavioral diagrams, activity, sequence, state machine and use case diagrams allow the modeling of the inputs, outputs, sequences, and conditions for coordinating the behaviors. To properly describe the security-related concepts (such as the vulnerabilities related to an operation) that could lead to discrete emergent security behaviors, both state machine and activity diagrams could be used. The State Machine diagram defines a set of concepts that can be used for modeling discrete behavior through finite state transition systems. While the activity diagram describes the behaviors and the constraints under which these behaviors may occur. For our work, we chose to extend the activity diagram mainly because the flexible link that could be defined to relate the actions/behaviors to the blocks owning those behaviors (possibility to relate the operation described in the activity diagram to the corresponding CS). The following is a summary of the SysML extensions to UML Activity diagrams. In addition, the SysML activity diagram allows the definition of nodes and constraints notes (possibility to extend these concept to express the SoSSecML security concepts).

Accordingly, we first describe below a part of each of the selected block definition and activity diagrams including the main concepts that we will reuse and/or extend (an additional detailed description of these diagrams could be fined in the SysML specification document [97]). Afterwards, we argue in the next section the use and extension of the SysML Block and Activity diagram concepts to define the SoSSecML concepts.

Figure I.2.6 shows a part of the block definition diagram including its main concepts:

- Block: Is the fundamental concept of this diagram. It defines a set of features to describe the system and its components, such as *Property* and *Operation*;
- Relationship: Illustrates the various relationships that *relate together* the blocks, like association, generalization and dependency;
- Port: Is a particular type of property used to specify *interaction points* between the blocks. Ports enable the flow of items between several blocks;
- Item Flow: Represents the information that *flows between* the blocks; These items could be specified using the *Flow Property*

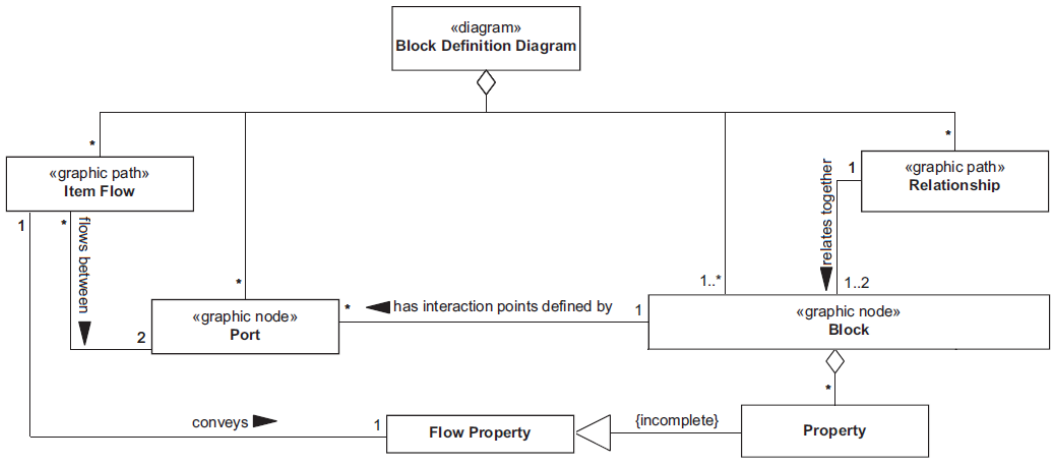


Fig. I.2.6 Part of the SysML block definition diagram MetaModel, adapted from [97]

Figure I.2.7 exhibits a part of the activity diagram including its main concepts:

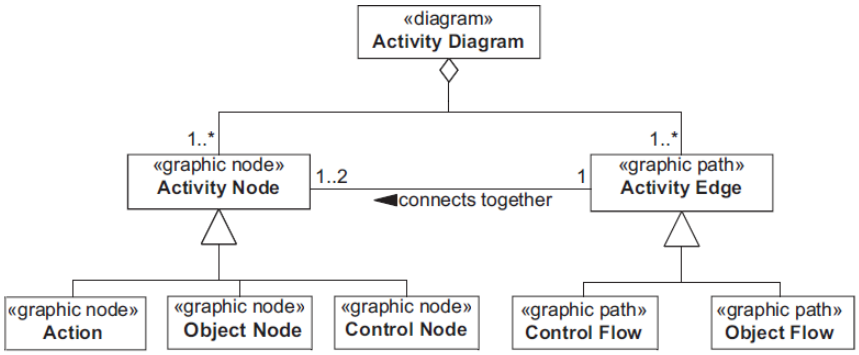


Fig. I.2.7 Part of the SysML activity diagram MetaModel, adapted from [97]

- Activity Node and Edge: The activity diagram is mainly *composed of Nodes connected together* through *Edges*. The *Activity Nodes* represents the individual steps in the behavior specified by the activity diagram, such as Object Node and Data Store Node. The *Activity Edges* are direct connections between two activity nodes along which tokens may flow, from the source to the target node;

- Action: It is the fundamental concept to specify a behavior. Each action may take a set of inputs and produce a set of outputs, both sets could be empty;

- Control Node: Is an activity node that manages the flow between other nodes in the activity diagram. Some instances of this node are: Initial node, final node, join node and decision node;

- Object Node: Is a kind of activity node that holds a value containing objects during the execution of a node;
- Object Flow: Is an activity edge that is traversed by objects that may hold values;
- Control Flow: Is a control edge that specifies the sequence of control nodes.

In addition, we employ a general basic concept, from the UML MetaModel, the *Constraint* concept, which represents an additional semantic information about a model element. It represents a restriction that must be satisfied.

We used the previously described concepts as basis to define our SoSSecML profile as detailed in the following subsection.

I.2.3.2 SoSSecML MetaModel

SoSSecML MetaModel specializes the SysML MetaModel for secure SoS architecture modeling. One one hand, SoSSecML extends the block diagram to enrich it with concepts specific to the structure of an SoS taking into account its specific characteristics. On the other hand, SoSSecML extends the activity diagram concepts to support the description of the SoS behavior in terms of functional interactions between the CSs and unknown security emergent behavior.

In our work we choose to address the *cascading attack* concern which is a crucial security problem with high-impact consequences in the context of SoS as detailed in the general introduction. Indeed, the collaboration between independent CS for the achievement of SoS global goal(s) results in emergent behaviors. In this context, small, known, unresolved and/or insignificant *Vulnerabilities* in each CS could be exploited and connected in an unknown way, to form a *cascading attack*.

It is important to recall that we focus in our contribution on the *unknown known vulnerabilities* category, as discussed in the general introduction. We consider that each CS defines a list of its own vulnerabilities (*known*). Since within the SoS none of the CSs have a global overview of the whole SoS by virtue of the managerial and operational independence characteristics and the absence of central authority, this list of vulnerabilities will remain *unknown* by other CS, hence the nomination *unknown known vulnerabilities*.

As explained in the previous section, a profile is mainly defined using the stereotype mechanism. Each new stereotype extends a concept or a relationship of the reference MetaModel. It should be linked to at least one existing Metaclass via an extension relation, or to an existing Stereotype via a generalization relationship.

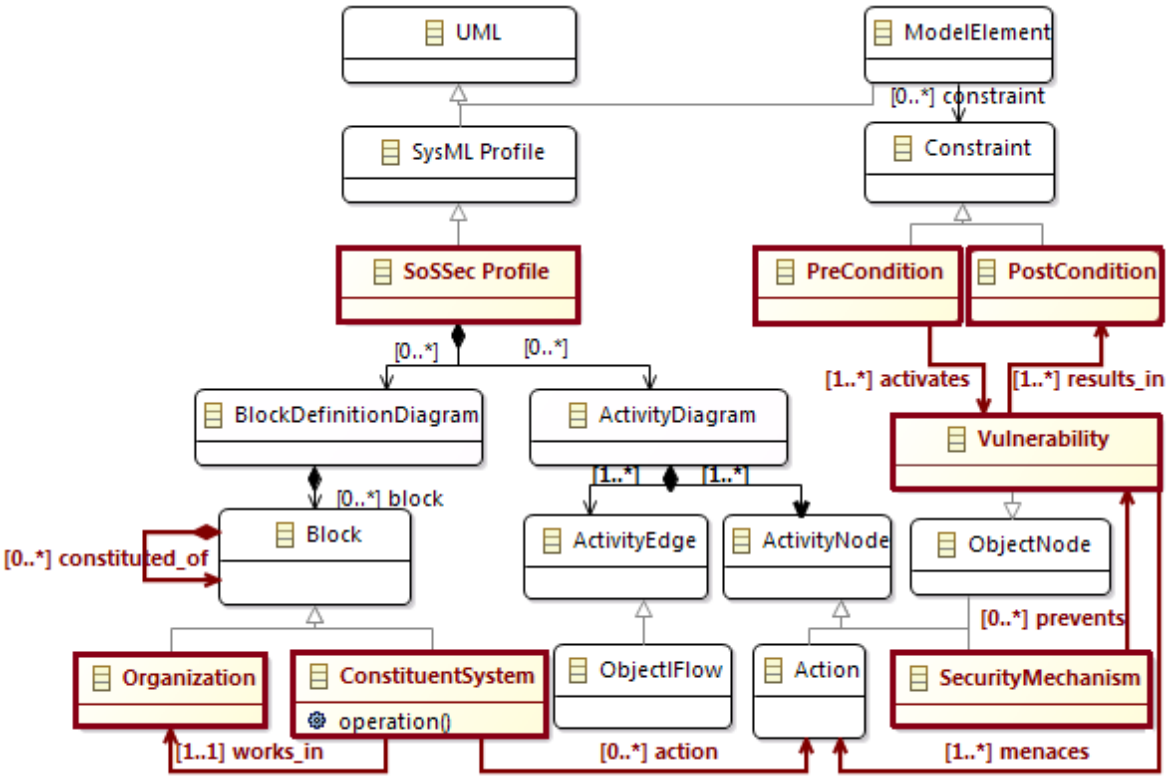


Fig. I.2.8 SoSSecML abstract syntax (MetaModel)

In figure I.2.8 we introduce our MetaModel. We note that the SysML MetaModel is one unified model with several views defined as diagrams. Therefore, we present our extensions in one figure including the basic SysML concepts in black and white and the added stereotypes in colors (or bold in black and white print). We now describe in details the introduced stereotypes and we recapitulate this description in table I.2.3 :

- ***ConstituentSystem*** as a generalization of a *Block*: we represented the SoS structure using the *Block* concept which can represent a system (hardware, software, data, person). We extended the *Block* by the *Constituent System* stereotype that represents the CSs as "black boxes" with several *operations/actions* on their interfaces, serving to model the interactions with other CSs.

  Indeed, what is crucial when modeling the SoS structure, is to properly describe the CSs boundaries to analyze the regular interactions between CSs as well as emergent behavior [41]. Mair evoked that "...the greatest leverage in system architecture is at the interfaces. The greatest dangers are also at the interfaces. There is nothing else to architect..." [75]. For these reasons, we model CS as "black-boxes" using *ConstituentSystem*. These CSs participate in the realization of the SoS objective(s)

through the *operations* representing the CS global functionalities defined on these interfaces.

It is important to mention that, according to the architects' need and desire, an extra level of details could be explored. The internal components of a CS could be modeled using the existing SysML concepts such as block, port and item flow. This is one of the advantages of building our DSL as a SysML extension. Nevertheless, we retained our work to the CS level seeing that this is a specificity in modeling SoS.

- ***constituted_of*** as an extension of the *compartment relationship*: A CS itself could be an SoS composed of several CSs. Therefore, we refine the *compartment relationship* of a *Block* to explicitly represent the fact that an Block could be composed of one or many Blocks, and by inheritance, a CS could be composed of one or many CSs.

- ***Organization*** as a generalization of a *Block*: The CSs in an SoS are operationally and managerially independent. They belong and are managed by different organizations/companies. To model these aspects, we introduced the *Organization* concept as an extension of the *Block* concept. The *Organization* is responsible of the functional as well as non functional aspects of the CS under its governance, specially security concerns.

- ***works_in*** as an extension of the *dependency* relationship: It serves mainly to tie a CS with the organization in which it works. A CS *works_in* one organization but an organization could manage many CSs.

- We reuse the SysML concepts Action and ObjectFlow to model CS operations and interactions: For each CS, only those operations described on the interface could be engaged to fulfill the SoS global goal. Due to managerial and operational independence characteristics, none of the CSs has a global overview of the SoS. With respect to this idea, there should not be a unified diagram that describe all the interactions within the SoS. Consequently, we consider that the operations of each CS are described in a different activity diagram using the existing Action concept and ObjectFlow relationship of the SysML activity diagram.

- ***Vulnerability*** as an extension of a *ObjectNode*: Since security is a behavioral aspect, we enriched the activity diagrams with security concepts to support ulterior analysis of security issues, in particular the possible sequence of triggered vulnerabilities leading to large attacks. We added the concept *Vulnerability* to represent the back doors or weaknesses in a CS. These vulnerabilities are assigned to the operations/functionalities

on a CS interface, therefore they could be connected through the CS interactions to achieve the SoS global goal.

- ***menaces*** as an extension of the *dependency* relationship: It is used to assign one or several *vulnerability* to an *Operation/Action*.

- ***PreCondition*** as a generalization of a *Constraint*: It defines the condition which activates/triggers a vulnerability. A vulnerability may have one or many pre-conditions. For example, for an SQL injection vulnerability that consists of a code injection to insert and run nefarious SQL statements on a database, the pre-conditions are the following: "user input is incorrectly filtered for string literal escape characters embedded in the SQL statements" or "user input is not strongly typed and unexpectedly executed".

- ***activates*** as an extension of the *dependency* relationship: To make the link between a *Vulnerability* and a *PreCondition* that *activates* this vulnerability. A Vulnerability could be activated by one or many PreConditions, but each PreCondition activates one vulnerability.

- ***PostCondition*** as a generalization of a *Constraint*: It defines the condition which results from an activated/triggered vulnerability. A vulnerability may have one or many post-conditions. For example, for an SQL injection vulnerability, the post-conditions are "information disclosure" and/or "total shutdown of the affected resource".

- ***results_in*** as an extension of the *dependency* relationship: To make the link between a *Vulnerability* and a *PostCondition* in which it *results_in*. A Vulnerability results_in in one or many PreCondition, but each PreCondition results from one vulnerability.

- ***SecurityMechanism*** as a generalization of an *Action*: To represent any possible mechanism defined to detect, prevent, or recover from a security problem. For example, "performing white list input validation" is a security mechanism to prevent an SQL injection by detecting unauthorized input before it is passes to the SQL query; and "Enforcing Least Privilege" is a mechanism that minimizes the damage of an SQL injection by minimizing the privilege assigned to the database in its environment.

- ***prevents*** as an extension of the *dependency* relationship: It sets the link between the *SecurityMechanism* and the *Vulnerability* that it *prevents*. A SecurityMechanism prevents one or many vulnerabilities and a vulnerability could be prevented by one or many mechanisms.

Table I.2.3 SoSSecML stereotypes

Added stereotype	SysML concept	Description
ConstituentSystem	Block	to model a CS as a black box with operations/functionalities on its interface
constituted_of relationship	compartment	to exhibit the fact that a CS itself could be an SoS composed of several CSs
Organization	Block	represent the company to which a CS belongs, to illustrate the managerial independence characteristic of a CS
works_in relationship	dependency	to tie a CS under its organization
-	Action	to represent the operations defined on a CS interface
-	ObjectFlow	to represent the interactions between CSs
Vulnerability	ObjectNode	to model the weaknesses related to a CS operation
menaces relationship	dependency	to link the vulnerability to the corresponding operation
PreCondition	Condition	to define the condition(s) that activates a vulnerability
activates relationship	dependency	to link each pre-condition to the correspondent vulnerability
PostCondition	Condition	to define the result(s) of an activated vulnerability
results_in relationship	dependency	to link each post-condition to the correspondent vulnerability
SecurityMechanism	Action	to model a security mechanism that could prevent, detect or recover from a security problem
prevents relationship	dependency	to associate the security mechanism to the vulnerability it helps preventing

By introducing these concepts that define the SoSSecML MetaModel (abstract syntax), we offer the necessary basics to model secure SoS architectures taking into account the SoS specific characteristics, in view of the analysis and discovery of unknown cascading security attacks. Actually, SoSSecML handles the modeling of the SoS structure taking into account its specific characteristics, dealing by that the first problem stated in the general introduction

chapter. Indeed, SoSSecML addresses the *managerial* and *operational independence* characteristics of an SoS by separately modeling each CS structure, the organization to which it belongs as well as its interface including the operations/functionalities and the interactions with other CSs. It also respects the *geographic distribution* characteristic by considering that the CSs are physically dispersed and only exchange data (not energy or mass).

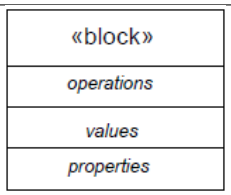
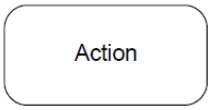
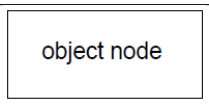
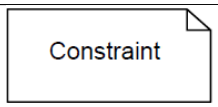
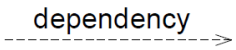
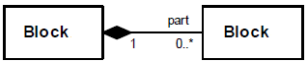
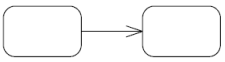
Moreover, the essential contribution is that SoSSecML addresses the aspects of the SoS *emergent behavior* through the modeling, which enables further analysis of security-related concepts. We associated these concepts to the CSs operations employed during the CSs interactions to achieve the SoS global goal. By introducing these security concepts (vulnerability, pre and post conditions, security mechanism) and their corresponding relationships, we enrich the behavioral model of the SoS with meaningful information that will serve, throughout the execution of the models, to detect the possible unknown sequence of activated vulnerabilities (cascading attacks) resulting from CSs interactions. In fact, the behavioral diagram models the interactions of the CS operations described on the interfaces, each operation along with the possible vulnerabilities that could be triggered once the operation is in use/executed. The description of these vulnerabilities is enriched using the pre-condition concept representing the reasons which may trigger the vulnerability; as well as the post-conditions representing the results of a triggered vulnerability. Therefore, when the described architecture is executed and the interactions are simulated, the vulnerabilities related to the executed CS operations could be triggered if their pre-conditions are met. In addition, the possible unknown sequence of triggered vulnerabilities could be discovered using a matching mechanism (that we detail in the analysis chapter). By modeling these security concepts we are enriching the SoS architecture models to allow, later on during the execution and analysis phase, the discovery of the unknown security cascading attacks emergent behaviors through the mechanisms we are proposing in the next part of this manuscript. Consequently, SoSSecML and its tools address the SoS security at the architecture phase of the SoS engineering life cycle, offering by that a solution to the second problem stated in the general introduction chapter.

Table I.2.3 recapitulates SoSSecML stereotypes, the SysML concepts on top of which we build these stereotypes and the newly defined concepts descriptions.

I.2.4 SoSSecML concrete syntax

As previously mentioned, defining a DSL encompasses corresponding one or many concrete syntax(es) to its abstract syntax. The concrete syntax is the actual human-readable notation used for the presentation and visualization of the models. It is a set of textual or visual notations that facilitates the language presentation and construction. Keeping the abstract

Table I.2.4 Part of the SysML - SoSSecML concrete syntax, modified from [97]

SysML - SoSSecML Node	Concrete Syntax
Block - Constituent System - Organization	
Action - Security mechanism	
Object Node - Vulnerability	
Constrain - Pre/Post-condition	
dependency relationship - works_in - menaces - activates - results_in - prevents	
compartment relationship - constituted_of	
object flow relationship	
generalization relationship	

syntax separate and independent from its concrete syntax(es) allows a modeling language to have several different concrete representations for diverse contexts or viewpoints [115].

The concrete syntax could be defined in a textual manner or in a visual/graphical manner. In a textual syntax the models are described using structured textual forms/keywords, whereas in a visual syntax the models are represented through diagrammatically forms/icons such as visualizing the nodes and edges notations through rectangle and arrow figures. For our SoSSecML concrete syntax, we choose to use a graphical approach. Actually, to benefit from the DSL extension mechanism, we reused the SysML visual notations to represent the new concepts that we introduced to the abstract syntax.

We present in table I.2.4 a part of the SoSSecML concrete syntax inherited from the SysML concrete syntax.

I.2.5 SoSSecML modeling tool

As discussed in section 1, the definition/extension of a DSL implies the definition of its corresponding set of tools, primarily a graphical editor and a code generator. MDE offers automated mechanisms and MetaTools such as the Eclipse Modeling Framework (EMF), the Graphical Modeling Framework (GMF) and exchange standards like the XML Metadata Interchange (XMI), to support and automate the definition of the modeling language tools, reducing by that the tool development time and effort. The main advantages of defining a tool following MDE are to make it standardized, modifiable, extensible and reusable.

I.2.5.1 SoSSec modeling tool construction process

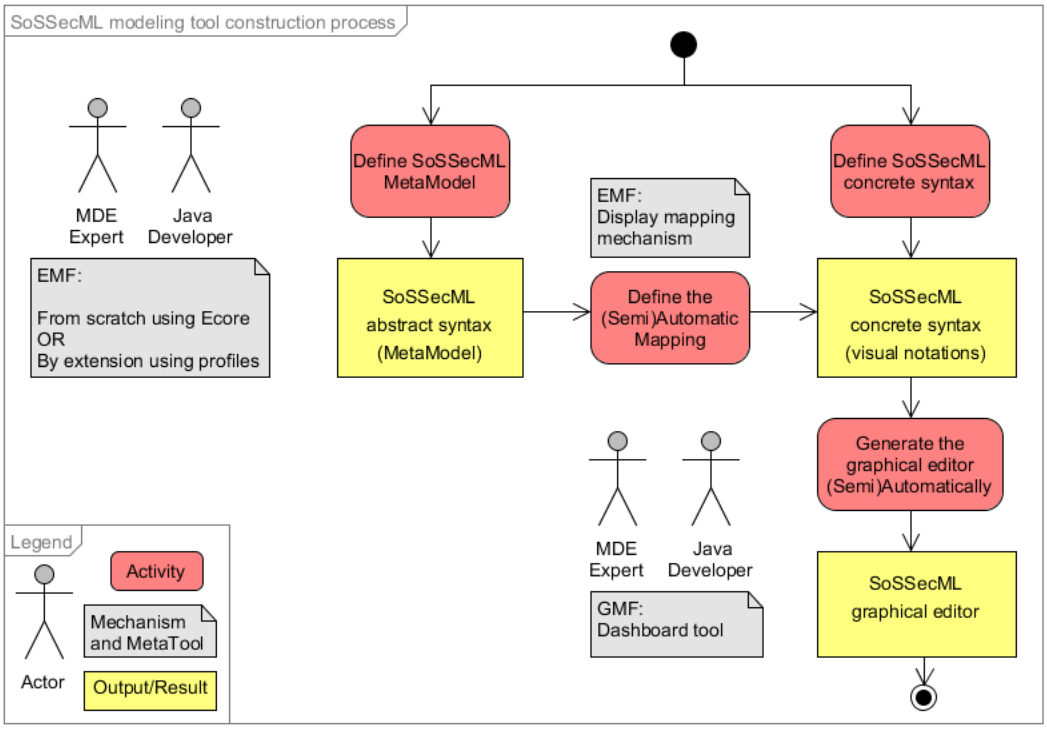


Fig. I.2.9 SoSSec modeling tools construction process

We took advantage of the MDE mechanisms and used the EMF to define a software modeling tool for SoSSecML. We present in figure I.2.9 the process that we followed to build the SoSSecML visual graphical editor. This process describes the activities, artifacts and MetaTools used to define the graphical editor. In fact, a *Java Developer* that might be assisted by an *MDE Expert*, uses the Eclipse Modeling Framework to *Define the SoSSecML MetaModel/abstract syntax* and its *concrete syntax* as well as the *(Semi)Automatic Mapping*

between them using the *EMF display mapping mechanism*/transformation rules. Having the concrete syntax, the same actors use the *GMF Dashboard tool*, presented in what follows and in figure I.2.11, to *Semi(Automatically) generate the graphical editor*.

As a first proof-of-concept of our SoSSecML abstract and concrete syntaxes, our Java developer - M. Zi Yang Pang[3] - developed a simple visual modeling tool, designed from scratch using EMF as described in the following sub-section. We used this graphical editor to model the secure architecture of a simple smart campus SoS [54], which helped us better discern and investigate our concepts and improve them. Afterwards, to further benefit from MDE extension mechanism and produce a powerful SoSSecML graphical editor that could be eventually brought to an industrial level, we extended one of the well-known SysML graphical editors: The Papyrus tool, as described in the later sub-section.

I.2.5.2 SoSSecML graphical editor based on EMF

The Eclipse Modeling Framework[4] (EMF) is an open source environment and Eclipse plug-in that supports MDE mechanisms for model design, transformations, interoperability and code generation. EMF provides the fundamental infrastructure and components for developing textual and visual tools in Eclipse. The EMF MetaMetaModel is Ecore: It is a general model from which any MetaModel can be defined. Ecore allows the definition of DSLs using meta-concepts such as *EClass*, *EAttribute*, *EDataType* and *EReference* and their corresponding relations as partially shown in figure I.2.10.

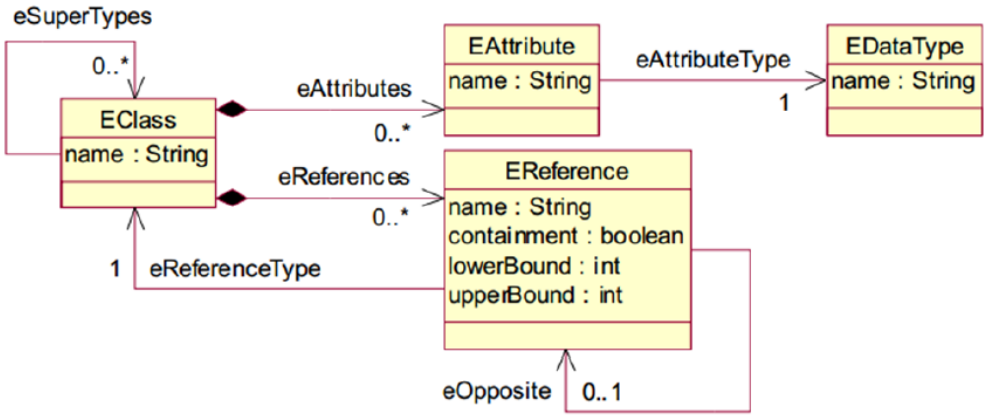


Fig. I.2.10 Part of the Ecore MetaMetaModel in EMF from [80]

[3]M. Pang was a master student at the University of Adelaide, Australia. He implemented the SoSSec graphical editor in the context of his three month master internship realized under the supervision of Mme. EL HACHEM

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

We specified our first version of SoSSecML abstract syntax (MetaModel) using Ecore by describing its classes using the *EClass* element (that owns attributes and relations with other classes), its attributes with the *EAttribute* classes, its types with the *EDataType* and the relations between classes with the *EReference* classes. This Ecore model is then used within the Eclipse Graphical Modeling Framework (GMF) to generate (parts of) the graphical editor. As shown in figure I.2.11, GMF provides the fundamental infrastructure and components for developing visual models and modeling surfaces in Eclipse. Having the SoSSecML MetaModel (Domain model-Ecore), GMF allows the definition of diagrams (Diagram Editor) based on a mapping between the abstract and the concrete syntaxes and the (semi)automatic generation of the corresponding visual graphical editor (Tooling Def). This graphical editor with a smart campus energy management example were presented in our paper [54].

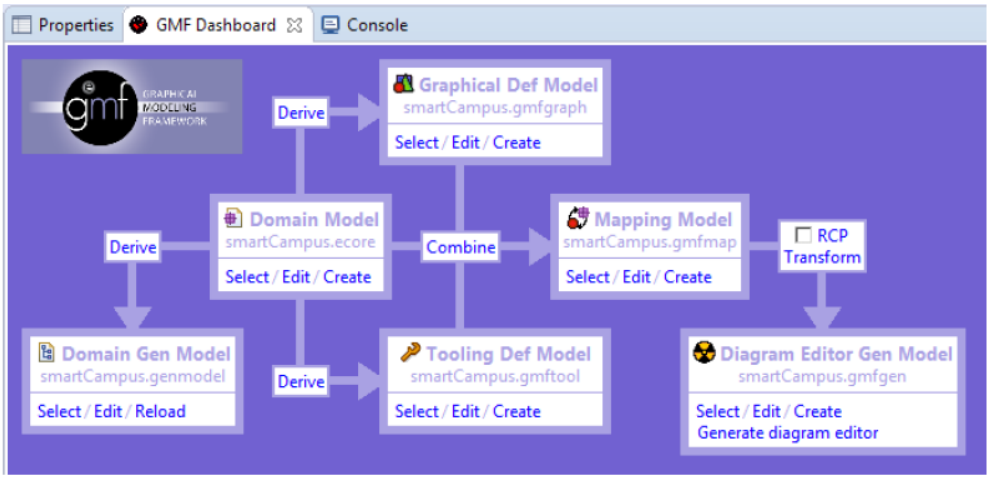


Fig. I.2.11 GMF Dashboard for visual modeling tools generation

I.2.5.3 SoSSecML graphical editor as a Papyrus extension

For this more complete and efficient version of SoSSecML visual modeling tool we extended the Papyrus[5] tool. Papyrus is an open source project based on Eclipse to provide an integrated environment for editing UML and SysML models. It is actively used by the software engineering community and properly documented and maintained by the CEA commission[6] and supported by active forums. Papyrus provides support for SysML and MDE such as diagram editors (GMF-based or not) and other MDE tools (profiling mechanisms), as well as advanced facilities for customizing the tool itself. It can be easily extended to define DLSs and edit EMF models reducing by that the development time and effort. Once the MetaModel

[5]https://eclipse.org/papyrus/
[6]http://www-list.cea.fr/en/

profile is defined in Papyrus, the extensions are automatically mapped into the tool and can be directly applied to design the models.

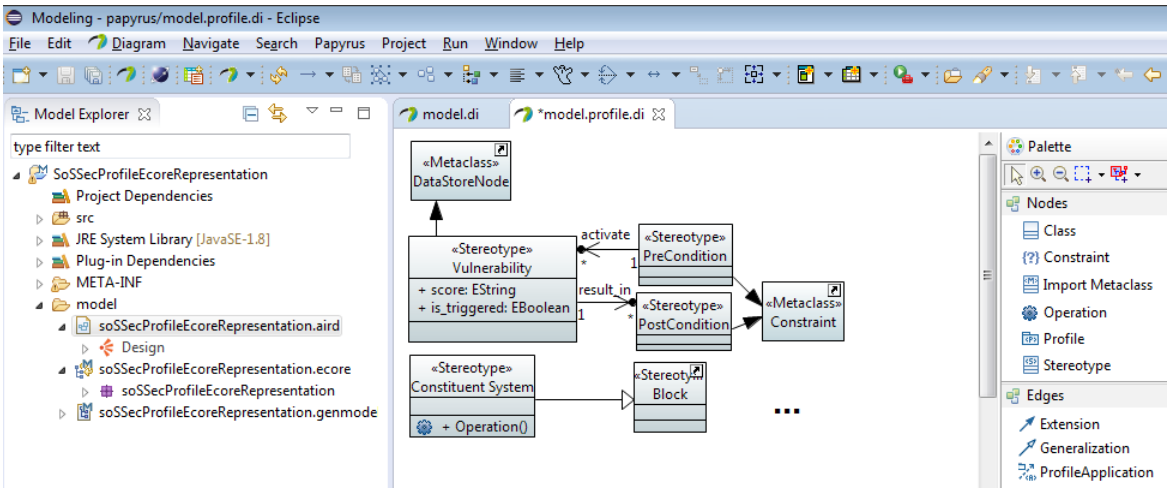


Fig. I.2.12 Part of the SoSSecML MetaModel implemented as Papyrus profile

Our Java developer (M. Pang) implemented the SoSSecML profile with the corresponding stereotypes using the MDE profiling mechanism already integrated in Papyrus as shown in figure I.2.12. Having our MetaModel defined as a SysML profile, Papyrus automatically maps the introduced concepts into the graphical editor which allows the designer to use them in term of stereotype concepts applied on the SysML model elements as shown in figure I.2.13.

We note that for implementation reasons, the *DataStoreNode* was used in the Papyrus activity diagram instead of the *ObjectNode* without any impact on the semantics of the latter node. Indeed, we are using Papyrus version 1.1.2[7] and to the best of our knowledge, the Object nodes can not be displayed in Papyrus activity diagrams[8,9]. Therefore we used the *DataStoreNode* which is an inheritance of the *ObjectNode* that serves to hold object tokens persistently while its activity is executing [98] (very similar to the definition of an object node, cf. section I.2.2.1).

By defining the SoSSecML graphical editor as a Papyrus extension, we offer a strong tool which allows the use of our newly introduced concepts besides those of SysML diagrams. Furthermore, Papyrus allows re-generation of the tool, whenever our language evolve, to incorporate the new features in an effortless, non-expensive way. Consequently, this version of the SoSSecML tool published in [55] has the following advantages: Inheriting SysML

[7] https://projects.eclipse.org/projects/modeling.mdt.papyrus
[8] https://www.eclipse.org/forums/index.php/t/1083704/
[9] https://www.eclipse.org/forums/index.php/t/1079866/

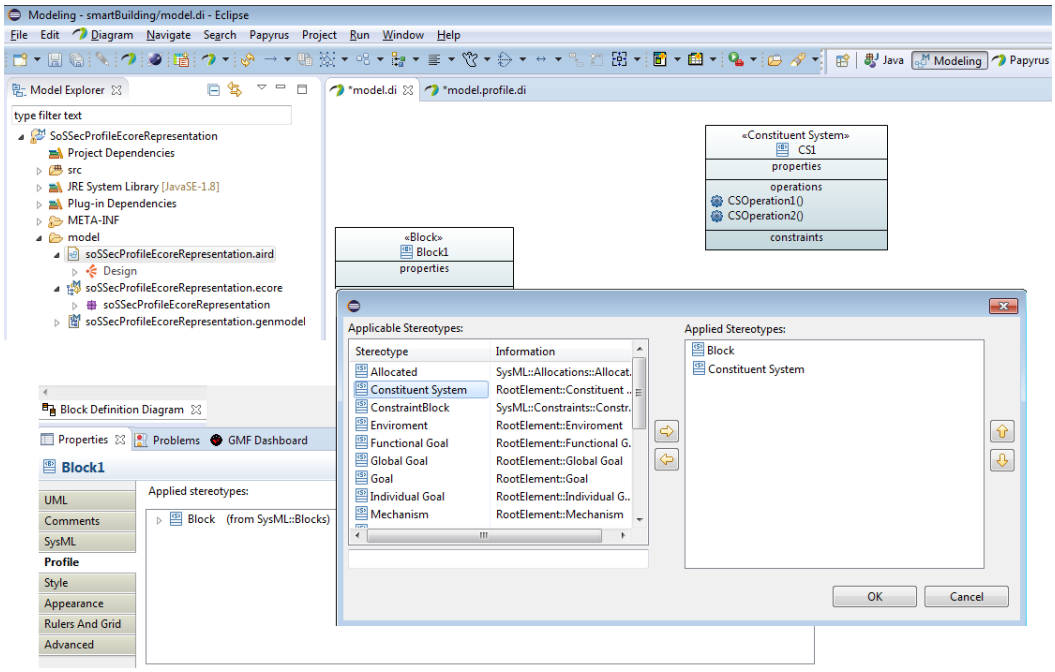


Fig. I.2.13 SoSSecML graphical editor defined as Papyrus extension

concrete syntax, offering a portable, customizable and extensible graphical editor, all this while taking advantage of the profile extension mechanism.

I.2.6 SoSSec utilization process - Modeling phase

To complete our SoSSec method we propose in figure I.2.14 an utilization process (published in [53]) that guides the use of the proposed language and tools to model secure SoS architecture. This process describes the activities that should be followed when using SoSSecML to model the secure architecture of an SoS, along with the corresponding actors and required tools.

In the modeling phase of our SoSSec utilization process, for each CS, different software and security architecture teams belonging to different organizations, model separately the structural, behavioral and security aspects of the CS for which they are in charge, using the SoSSecML graphical editor, guaranteeing the *managerial and operational independence* characteristics. The architects define the CS, the organization to which it belongs, the operations on its interface representing the direct interactions with other CSs to participate in the achievement of the global goal, and a list of possible vulnerabilities related to these operations along with their pre and post-conditions.

To model each CS, we further detail the SoSSec Utilization process in what follows.

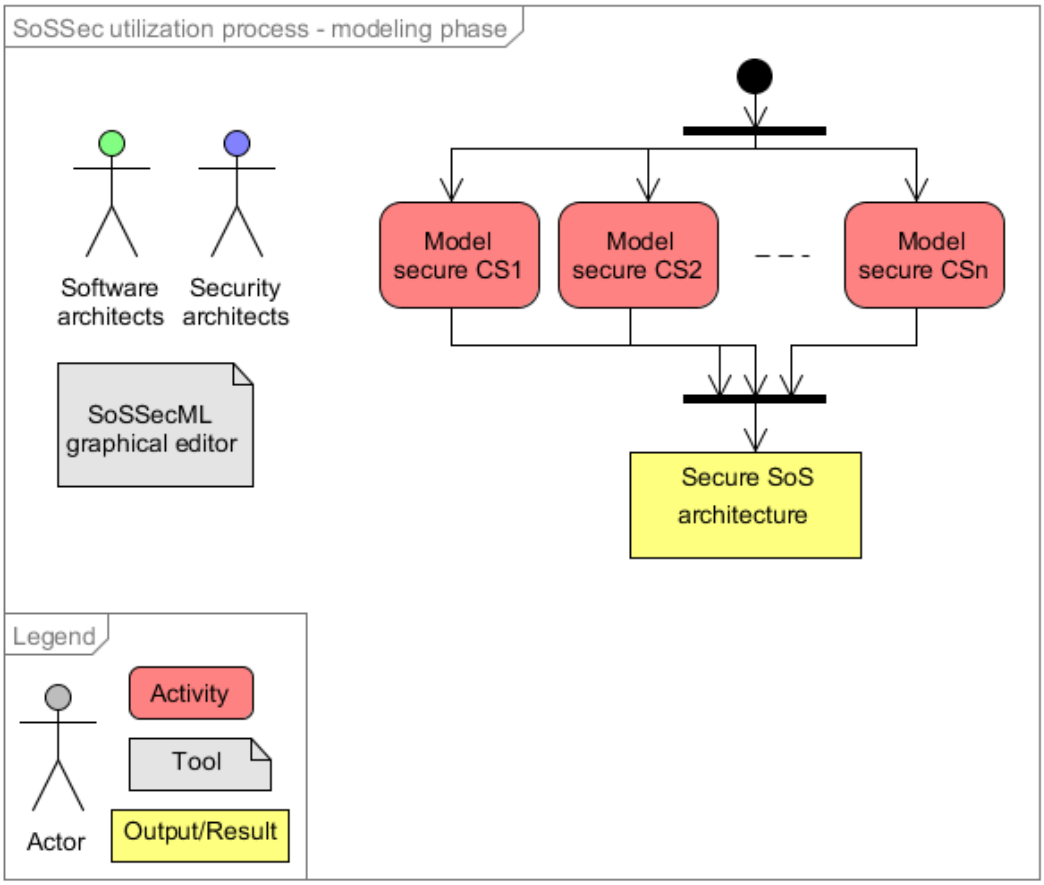


Fig. I.2.14 SoSSec utilization process - Modeling phase

As shown in figure I.2.15, to model each CS in the SoS, the software and security architects first define the needed functional and non functional requirements. Then, they model the structure and behavior of each individual CS, using SoSSecML structural and behavioral diagrams and in respect to the defined requirements. Moreover, the architects define a list of possible vulnerabilities with their pre and post-conditions related to each CS operation based on their expertise and/or information extracted from vulnerability databases and dictionaries. As detailed in figure I.3.9, the the software and security architects of each CS define the lists of vulnerabilities for each CS operation based on their security expertises, the CS component and operation details and their functional and security guides/instructions, as well as the security and attack publications and news. Having the vulnerabilities, the architects can use CVE or any other vulnerabilities database to collect detailed information about the vulnerability, the reasons/open doors that lead to this vulnerability and the results/damages coming from it. Afterwards, the architects can analyze these information and their causal relations to extract a list of pre-conditions representing all the possible conditions that

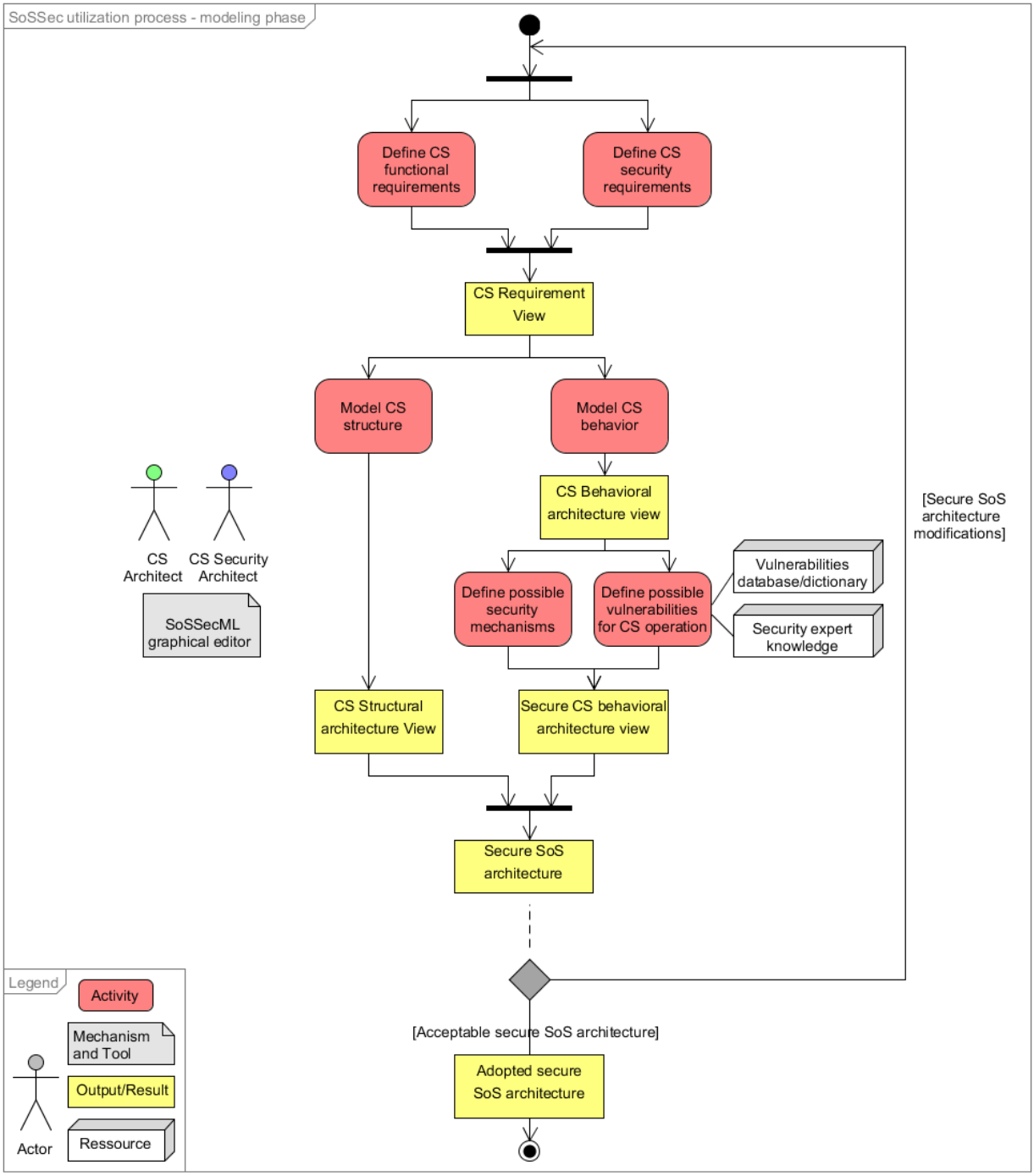


Fig. I.2.15 SoSSec utilization process - Detailed modeling phase

activate the vulnerability and a list of post-conditions representing all the possible results of the activation of this vulnerability.

Having the structural and behavioral views of each CS, the different architectures are combined to form the SoS secure architecture. We note that all CS architectures are modeled

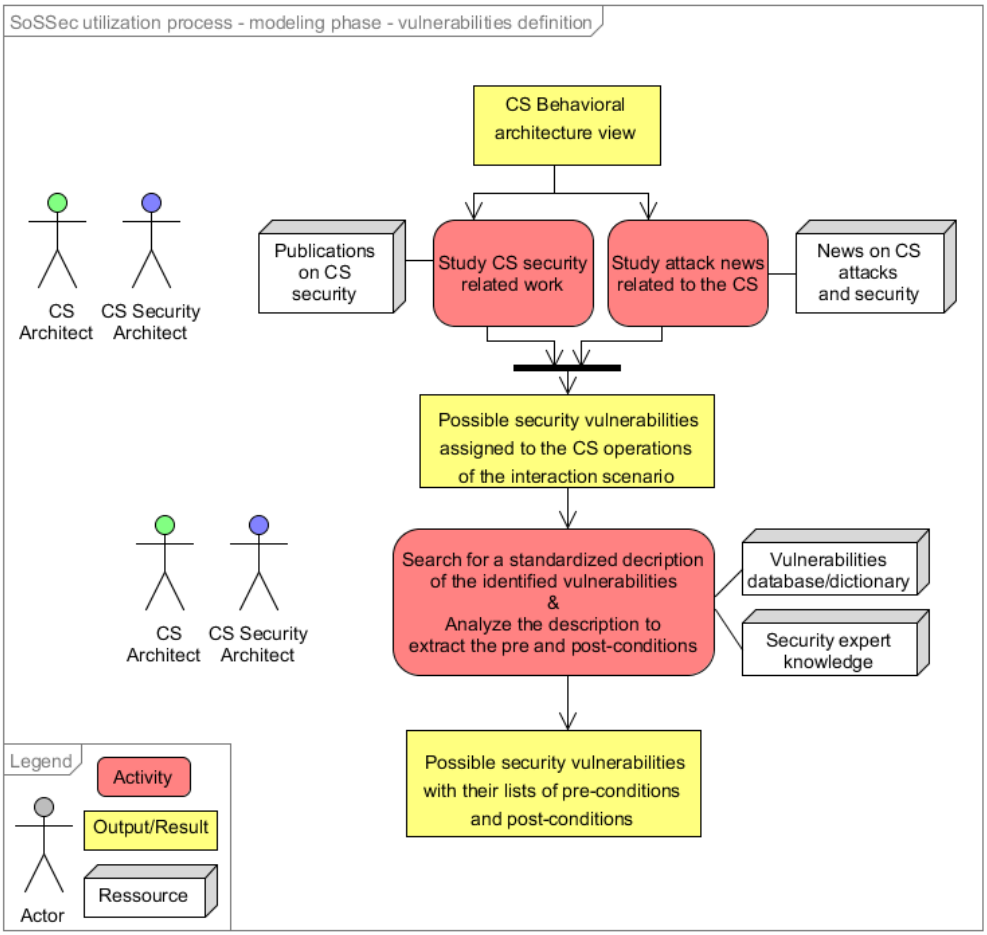


Fig. I.2.16 SoSSec utilization process - modeling phase - vulnerabilities definition

using SoSSecML graphical editor to form the SoS secure architecture; we do not deal with the composition of models [122]. This process will be completed with its analysis phase in the second chapter of the part II, figure II.2.12.

Moreover, it is important to mention that one of the advantages of extending the SysML language and its Papyrus graphical editor is that other views of the SoS architecture could be modeled using SoSSecML and its tool. For example, the requirement diagram to represent the CS requirements and capabilities to achieve the global and individual goals, and the relationships of these requirements (such as derive, satisfy, verify, refine) to each other and to other model elements. In addition, the internal block diagram could be used to model the internal components of each CS.

I.2.7 Conclusion

To conclude, we presented in this chapter the abstract and concrete syntaxes of our MDE-based DSL called SoSSecML. As we concluded in the Related Work chapter I.1, we chose to extend a DSL, and for the reasons invoked there, we focused on SySML. Therefore, the SoSSecML MetaModel is defined as a profile extending the SysML MetaModel, mainly by the following concepts: Constituent System, Organization, Operation, Vulnerability, Pre-Condition, Post-Condition and Security Mechanism; along with their corresponding relationships.

In addition to the SysML concepts that cover some basic aspects of SoS modeling such as the CSs and their internal components, through the block definition and block internal diagrams (as concluded in the previous chapter I.1.5), our extension allows the modeling of an SoS taking into account not only the SoS specific characteristics, but also security attacks related concepts. By modeling these concepts, we are offering a representation of the secure SoS which allows, at the simulation and analysis phase, some types of emergent behavior to manifest and therefore become analyzable (in particular the unknown security cascading attack).

Consequently, SoSSecML, its tools and utilization process answer the RQ1 by offering the SoSSecML extension of a modeling approach to model SoS structure and behavior with a particular focus on security aspects to allow the prediction of unknown security emergent behaviors such as cascading attacks, early at the architecture level of the SoS development life cycle.

After presenting our language abstract and concrete syntaxes and its corresponding visual modeling tool (graphical editor) and utilization process, we present in the next chapter a real-life smart building that we use to illustrate our first contribution by modeling the smart building SoS, conforming to our DSL.

Chapter I.3

Case Study: Adelaide Health and Medical School (AHMS) smart building secure architecture modeling using the SoSSec method

I.3.1 Introduction

After presenting the modeling part of our SoSSec method including the SoSSecML DSL abstract and concrete syntaxes, its corresponding graphical editor (modeling tool), and the utilization process to be followed when using the method, we present in this chapter the application of the modeling part of our SoSSec method on a smart building SoS case study.

It is estimated that by 2050, 70% of the world's population will be living in cities [121]. Smart cities is an emerging paradigm of conceiving solutions to address urbanization challenges. Most of the smart city constituents can be classified as SoS [105] particularly smart buildings, which are considered as one of the basic blocks of smart cities. Building infrastructures and services are changing due to their CSs (e.g. automated systems, energy grids, emergency management) becoming more interconnected and smarter, nonetheless this leads to new security challenges and cyber-attacks such as privacy breaches, blackouts, buildings lock-down, fake emergency alerts, ransomware and DDoS attacks. These challenges should be taken into consideration when conceiving smart building solutions.

The Adelaide Health and Medical School[1] (AHMS) of the university of Adelaide in South Australia, is a real-world smart building. This building, opened for teaching in 2017,

[1] https://www.adelaide.edu.au/west-end/

costs AU$246 million and is considered the largest capital works' project in the history of the University. It is a first step on the road of the construction of several smart buildings in the context of the *smart campus master plan 2016-2035 project*[2]. This project is aligned with many other projects envisioned to make Adelaide a safe smart city [8].

In [20], the authors assume that using a new method to implement an existing system could be considered as a validation activity and provides useful information about the advantages and limitations of the new method. Whence, we use the AHMS smart building as a case study throughout this thesis, on which we apply our SoSSec method to prove its effectiveness in modeling and analyzing the secure architecture of this SoS.

In this chapter, we exemplify the SoSSec modeling part of the SoSSec method on the AHMS smart building SoS. Therefore, we use the proposed SoSSecML along with its graphical editor to model the secure software architecture of the AHMS. This work was accomplished in collaboration with the University of Adelaide[3], during a three month international mobility/research visit to the CREST[4] team and under the supervision of Prof. Ali BABAR[5]. The chapter is organized as follows: In section I.3.2, we discuss the current state of the smart buildings security. In section I.3.3, we present the protocol that we followed to elicit and conceive our AHMS case study, mainly the data collection phase. Section I.3.4 describes the AHMS smart building modeling phase: It includes the description of the smart building as an SoS with respect to Maier's OMGEE specific characteristics, the detailed SoSSec utilization process which describes the actors, tools and steps to follow to model an SoS secure architecture and the use of this process to model the secure architecture of the AHMS smart building. Section I.3.5 concludes the chapter.

I.3.2 Smart building security current state

SoS are open to cascading attacks with significant effects caused by small vulnerabilities remaining in each of their CSs and exploited due to CSs interactions (cf. general introduction). Smart buildings are one of the SoS application domains where security is crucial and cascading attacks may have destructive results. Smart buildings' *confidentiality* could be disrupted by unauthorized access to different CSs and their data, through unauthorized paths such as unauthorized access to the booking CS or to the access control management CS to reveal personal data related to resident like their presence, absence and important visitors. A building's *integrity* could be compromised if an attacker gained access or controlled the

[2]https://www.adelaide.edu.au/infrastructure/projects/current/masterplan/
[3]https://www.adelaide.edu.au/
[4]CREST: Center for Research on Software Engineering Software Technologies, http://crest-centre.net/
[5]https://malibabar.wordpress.com/about/

critical automated CSs of the building, for example by degrading or disrupting the operations of these CSs through direct interference or injection of malwares. The *availability* of a smart building services could be troubled by preventing the delivery of the building CS functionalities, for example hacking the heating CS would make the building inhospitable or damages its equipments, blocking the lifts and doors locks the occupants inside the building. Furthermore, many other attacks may target the building automated CSs[6], attacks such as interception (network sniffing), modification (man-in-the-middle attacks, alteration), interruption (denial of service, network flooding, redirection), fabrication (insert malformed messages, reply old messages), software (code injection, exploiting algorithm weakness, configuration mechanism abuse), physical (eavesdropping, micro-probing, component replacement). Table I.3.1 shows some recent attack examples extracted from different smart building/home attack news.

Table I.3.1 Smart buildings/homes attack news and related security concerns

Attack	**Targeted CS**	**Results**
Ransomware attack on a fully booked hotel in Austrian Alps[7], 2017	Infiltration to the electronic key system through code injection	$1,800 ransom, hotel locked out of its own computer system, guests stranded in the lobby, occupant confusion and panic.
Distributed denial of service attack taking down the central heating system during winter in Finland[8], 2016	Hackers disabled the computers that were controlling the heating system	A temperature below freezing and a long-term disruption in heat caused both material damage and a need to relocate residents elsewhere.
Distributed denial of service against the Target Store in USA[9], 2014	Building management system	Unauthorized access to more than 100000 devices revealing customers data including 53 million email address and credit card information.
Google's Australian office building hack[10], 2013	Unauthorized access to the building management system	Possible denial of service and false alarms activation.

One of the famous smart building attack is that of a Google's building: Two researchers accessed remotely the Building Management System (BMS) through zero-day vulnerabil-

[6] http://www.automatedbuildings.com/

[7] https://www.nytimes.com/2017/01/30/world/europe/hotel-austria-bitcoin-ransom.html

[8] http://metropolitan.fi/entry/ddos-attack-halts-heating-in-finland-amidst-winter

[9] https://securityledger.com/2014/11/third-party-vendor-source-of-breach-at-home-depot/

[10] http://www.smh.com.au/technology/technology-news/australian-google-office-building-hacked-20130506-2j416.html

ities[11] in the *Niagara framework* developed by the Tridium incorporation[12] (Niagara is a universal software framework that owns more than 25000 systems connected to the Internet and is used to build web-based applications to access, automate and control smart devices in a real-time over the Internet or through local network). Once in the system, the researchers found a privilege escalation bug that allowed them to access the configuration data including usernames and passwords. Then, the researchers decoded the administrator's password "anyoneguess". Having these authentication information, they were able to login to the systems from operators' accounts and control the CSs managed by these operators. The researchers claim that they were able to manage a large number of panels and press buttons like *active overrides* and *active alarms*.

This is an example of a cascading attack that starts from a simple vulnerability in one of the smart building CS. Many other similar vulnerabilities may exist in these CSs due to several reasons such as:

- Building CSs are connected to the Internet without the necessary security protection: The current number of Internet-connected buildings exceed 50000, among them 2000 reportedly without password protection[13];
- Smart buildings contain devices with hard-coded passwords that cannot be changed and back-doors not taken into account by vendors, for example sensors and thermostats that cannot reboot for software updates, and replacing these components is a difficult and expensive task for the stakeholders;
- Smart building software systems may not run the latest anti-virus software thus leading to potential vulnerabilities from malware introduced over the network or from infected media;
- Many implemented systems and devices of the smart building lack of security testing: Most of them are without any simple checklist for encryption, authentication or authorization;
- Numerous vendors do not provide all security documentation and timely response to security incidents such as patching and fixing security issues as soon as they are discovered.

[11] A zero day vulnerability is a hidden vulnerability that was discovered/known but exploited by attackers before being patched/solved

[12] https://www.tridium.com/

[13] http://www.bbc.com/news/technology-35746649

Despite its importance, security is barely taken into account in the smart building development life cycle, certainly not at the architecture level. In a recent systematic mapping study [45] on quality attributes for ambient assisted living software systems (e.g. smart buildings CSs), security has been identified as the most significant quality attribute based on the fact that the highest percentage (51.8%) of selected studies referred to security.

Moreover, a survey[14] carried out by Electrical Constructors' Association (ECA) and Scottish trade body SELECT states that roughly 40% of smart buildings do not take any fundamental security measures. In the case of AHMS, security design and analysis seems to not have fully been taken into consideration, opening by that the door to possible similar types of attacks.

I.3.3 AHMS case study planning protocol

To carry out our case study, we followed the planning protocol template presented in [20]. This protocol template improves the accuracy of the case study by ensuring that the data collection and procedures answer rigorously the defined goals. It identifies several steps that ensure a consisting planning process. Conforming to these steps we present in the following subsections the AHMS case study planning protocol we implemented.

Background

The background's main aim is to define the research questions to be addressed by the case study. For our work, after a first meeting with the energy management group of the University of Adelaide, notably Ms. Libby DOWLING [15] and Mr. Vikram KENJLE [16], where we presented our SoSSec method (encompassing the language, processes and tools) and discussed its possible applicability on a smart building SoS, in particular the AHMS case study, we agreed that the case study should answer the following questions:

- How can we use the SoSSec method to model and analyze the secure architecture of the most recent smart building of the university (the AHMS building)?
- How can the analysis results help improving the secure architecture of the future smart buildings?

[14] http://www.smartbuildingsmagazine.com/news/smart-buildings-remain-vulnerable-to-cyber-attacks

[15] Environmental Project Officer, Infrastructure Branch - Strategic Portfolio Management, Division of University Operations, The University of Adelaide

[16] University Energy Manager, Infrastructure Branch - Strategic Portfolio Management, Division of University Operations, The University of Adelaide

Case study research design

The central components of a case study design are the definition of the object/aim of the study, and the identification of the theoretical propositions derived from each research question.

For our case study, the aim behind using SoSSec to model and analyze the AHMS smart building is to help the architecture and security teams achieve the following goals:

1. Better understanding of the AHMS software architecture and the interactions among its different systems: This goal could be achieved by describing the AHMS as an SoS with the corresponding CSs and interactions, and modeling the AHMS structural and behavioral aspects conforming to the SoSSec modeling language (SoSSecML) and using its graphical editor;

2. Introduce security knowledge to the AHMS smart building architecture: By defining the possible vulnerabilities/weaknesses related to each CS of the AHMS smart building, and introducing these vulnerabilities and their details in the AHMS architecture;

3. Have a feedback on possible security issues related to this architecture: By analyzing the AHMS secure architecture and discovering the emergent related cascading attacks.

In addition, the design should identify the type of the case and its analysis unit. Referring to [127]:

- A *case*: Represents the interesting topic of the study. Cases could be *single cases* if the cases seems to represent a critical test to existing theory; or *multiple cases* if a "replication logic" is supposed to reveal support for the theoratical propositions;

- A *unit of analysis*: The actual source of information. It could be holistic or embedded designs. Holistic designs include a single unit of analysis to study the global nature of the phenomenon. Embedded designs include multiple units of analysis within a case.

Since we were conducting a single case study in a single company/organization to test our SoSSec method, our case study is a single case study. Additionally, our aim is to model and analyze the secure architecture of the case study, thus our case study is a holistic case.

Case selection

The criteria for selecting the organization and team with whom we collaborated to realize our case study were the following:

- A relatively realistic-scale case study in the context of smart cities;

- Ability to describe the target system as a System-of-Systems based on Mair's definition [75];
- Capability to collaborate with the target architecture and security teams on the following tasks:
  - Collect the needed data;
  - Understand the functional requirements of the system and its constituents;
  - Gather some possible security concerns from the security team if existing or define them if not;
  - Define a list of possible security vulnerabilities for each constituent system.

Knowing that several projects are under way to make Adelaide a safe intelligent/smart city, and considering the fact that designing and implementing a case study of smart city initiatives is an impractical task, it is more reasonable to design and implement solutions on a smaller but still realistic scale [8]. As it fulfills the listed criteria, a smart building, part of the smart campus master plan 2016-2035, aiming to provide an integrated strategy for the University's built environment in support of its strategic objectives, institutional values and international tertiary sector conditions, seems to be a satisfying case study to apply our SoSSec method.

Case study procedures and roles

In order to properly plan the requirements of our case study, such as the needed data and its analysis, we detail in what follows the procedures followed for each of these requirements (data collection plan, data analysis strategy). In addition, we clarified the roles of the participants in this AHMS case study as follows:

- The SoSSec research team (a Ph.D. student Ms. EL HACHEM supervised by Prof. BABAR, Prof. ANIORTE, Dr. CHIPRIANOV and a master student M. AL KHALIL[17] supervised by Ms. El HACHEM and Dr. CHIPRIANOV) are responsible of applying the SoSSec method to model and analyze the secure software architecture of one or several AHMS scenarios;
- The AHMS team (mainly the energy manager from the infrastructure branch: Mr. KENJLE) is responsible of offering the needed data as input for the case study and

[17] Tarek AL KHALIL, engineering student in Computers and Network Engineering at the Antonine University, Lebanon

evaluating the outcome of this study, its validity and usefulness for modeling the future smart buildings of the Adelaide university.

Data collection

1. To conduct this case study, we collected from the smart building architecture team the following set of data:

   - The global goal/mission of the AHMS smart building;
   - The constituent systems of the AHMS;
   - Functional requirements of the AHMS smart building as well as functional requirements of each of its CSs;
   - Some detailed information on the interactions among the AHMS CSs to achieve the smart building global goal/mission;
   - Non-functional security requirements of the AHMS smart building as well as non-functional security requirements of each of its CSs;
   - Details about the CSs such as the platforms they use and their software components, in order to define a possible list of vulnerabilities/weaknesses for the selected CSs;

   In addition to these data, we conducted a review to study existing publications/documentation on security issues or emergent behaviors targeting smart buildings along with the sources of those issues and behaviors.

2. The above described data was collected from two sources:

   - Meetings/Interviews: In addition to the first meeting with Ms.DOWLING and Mr. KENJLE, we had four meetings with Mr. KENJLE. In supplement, we exchanged with him several emails and documents;
   - Documentation study: Additionally, we studied existing publications such as [106][24] [116][120][104] and reports/press news such as [58][25][11][119] and the sources in table I.3.1 on the following topics:
     - Smart building systems considering the SoS characteristics;
     - Smart buildings security related concerns and attack news;

Data analysis

Based on [127] description, the collected data could be analyzed through three ways:

- Relaying on theoretical propositions: The focus on certain data and ignorance of other is guided by the theoretical proposition that has formed the design of the case study;
- Thinking about rival explanations: The original theoretical propositions could include some conflicting hypothesis. The collected data could possibly comprise some evidence about the possible "other influence";
- Developing a case description: Define a descriptive framework for organizing the case study.

In our work, since the method (language, utilization process and tools) were already defined, we adopted the first data analysis strategy because it seemed to be the most appropriate for our needs.

Plan to validity

The aim of this case study is to validate our SoSSec method by using it to model and analyze the secure architecture of an existing smart building. This could be considered as an exploratory case study to identify the effectiveness of the SoSSec method, its advantages and limitations. The results of the AHMS case study were discussed and analyzed in a meeting with the AHMS team that certifies the efficiency of the results and their utility in improving the security of the current AHMS building, as well as the prominent potential of the SoSSec method to improve the modeling and analysis of the secure software architecture of the future smart buildings of the Adelaide university.

Study limitations

The case study limitations are usually related to validity issues related to the study rather than the plan. Following are some of our AHMS case study limitations:

- Reliability: We collected the data from several meetings with a reduced number of specialists. We did not discuss the profound details with specific experts representing the different organizations in charge of the various AHMS constituent systems;
- Anonymity: In our case study, we excluded some descriptions to protect the anonymity;

- Case study selection criteria: The features used to select the AHMS case study research may present certain limitations in its usage for different application domains (such as the AHMS interaction scenario where the structure and interactions of the fire detection CS is subject of the Australian regulations about fire safety);
- Case evaluation: The case study and its results were validated with the AHMS team, but not confronted with security experts;
- Single case study: We conducted a single case study on one smart building SoS;
- Generalization: The generalization could be formalized or left for the reader who determines what can apply to his context. We note that by arguing that a smart building, in particular the AHMS, is an SoS with respect to the OMGEE specific characteristics, and by applying the SoSSec method on the AHMS without any need of refining it, we assume that it could be generalized to other application domains without complications.

Reporting

The results of this case study were reported in a document addressed to the AHMS team. In addition, the results will be published in one or more academic papers targeting international conferences and journals and in this thesis.

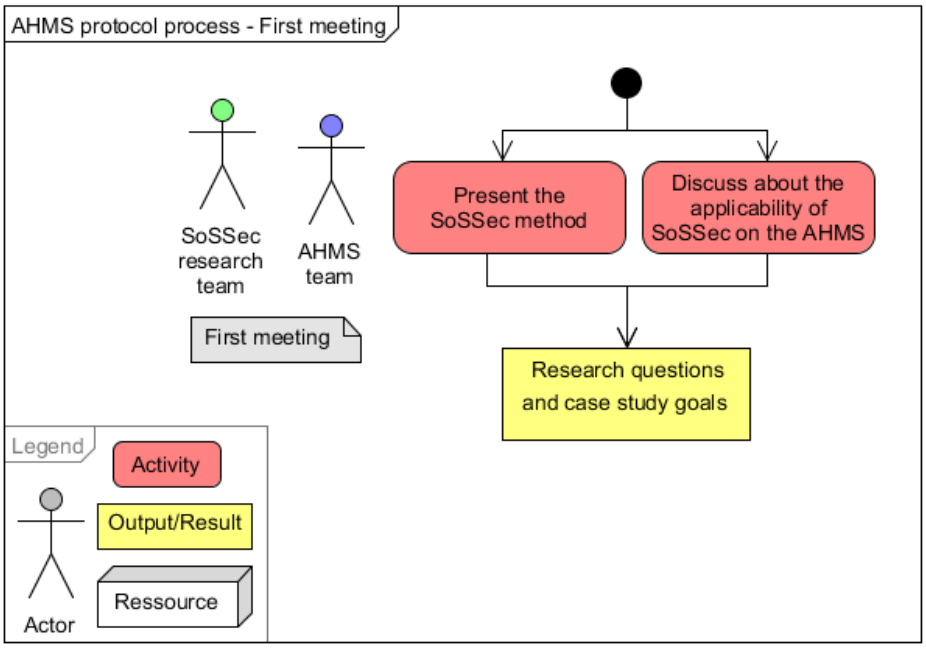


Fig. I.3.1 AHMS planning protocol - First meeting

Schedule

The case study was realized in a period of two months.

In what follows we describe the detailed process that we followed, with respect to the previous template steps, to realize our AHMS case study. We describe the actors/teams and their roles, as well as the different inputs and outputs of the AHMS case study with respect to the meetings and exchanges between the teams.

1. In the first meeting (figure I.3.1), the SoSSec team presented the SoSSec method (the language, processes and tools) and discussed with the AHMS team how their method could be of benefit for the AHMS building secure architecture modeling and analysis. The meeting ended with a list of research questions to be answered;

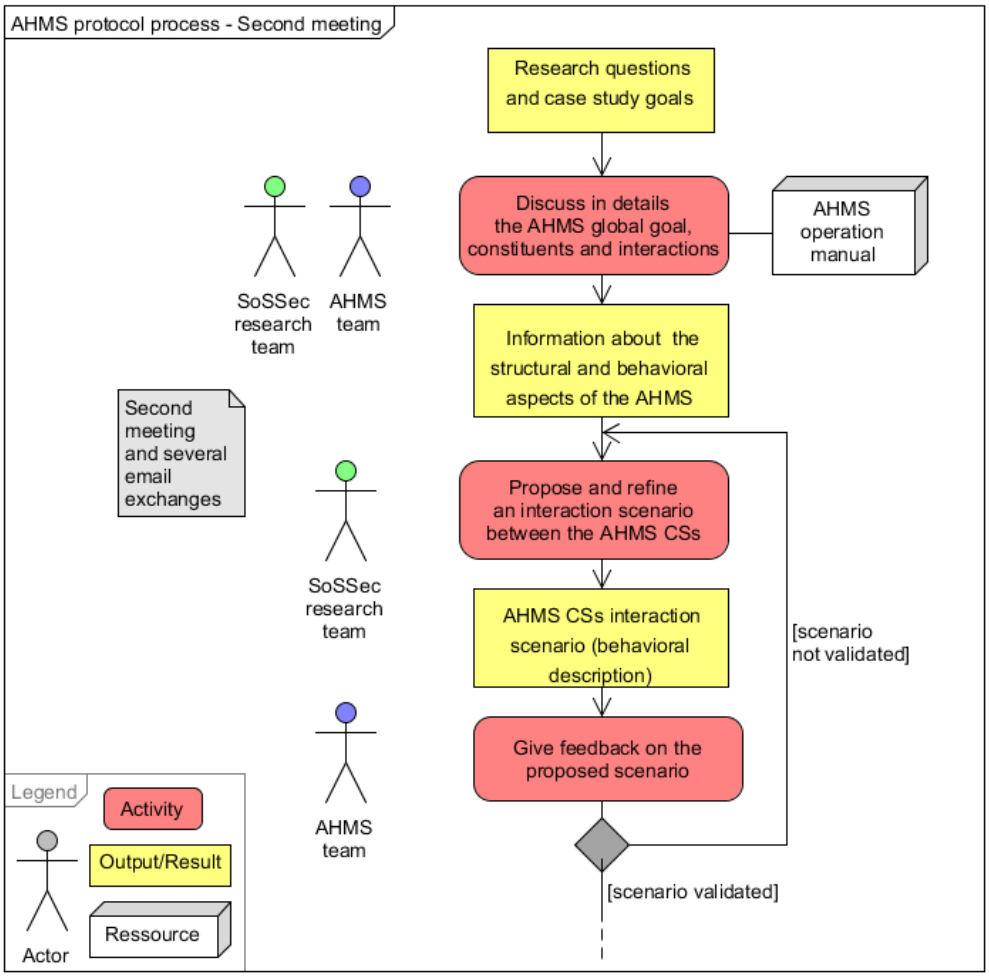


Fig. I.3.2 AHMS planning protocol - Second meeting

2. In a second meeting (figure I.3.2), both teams discussed more in details the AHMS possible global goals, constituent systems (CSs) and their components and the interac-

tions between these CSs to achieve the global goals. Having these information, and the AHMS operation manual, the SoSSec team proposed an interaction scenario with respect to one of the global goals. This scenario was refined several times following the AHMS team feedbacks (sent through emails), until it was validated by them;

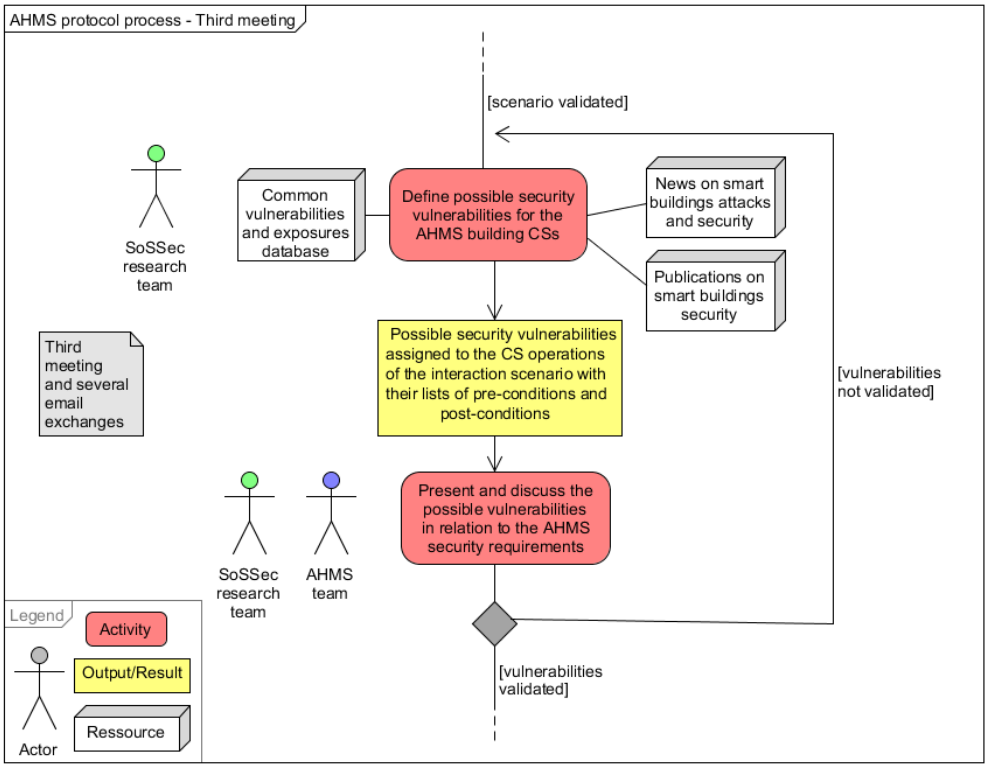


Fig. I.3.3 AHMS planning protocol - Third meeting

3. Having the structural and functional behavior of the AHMS, the SoSSec team defined the list of possible vulnerabilities with their pre and post-conditions for each CS operation. Afterwards, the SoSSec team presented these results and discussed them with the AHMS team in their third meeting (figure I.3.3). Several feedbacks and refinements took place until the vulnerabilities and their details were validated;

4. After collecting and analyzing the AHMS structural, functional and security data, the SoSSec team applied the SoSSec method to model the AHMS secure architecture. Once this architecture was validated by the AHMS team, the SoSSec team analyzed it with their method and suggested possible requirements to improve the AHMS smart building security (figure I.3.4). The results were presented and discussed with the AHMS team that founds the output valuable and the method very interesting for the modeling and analysis of the future buildings of the University of Adelaide.

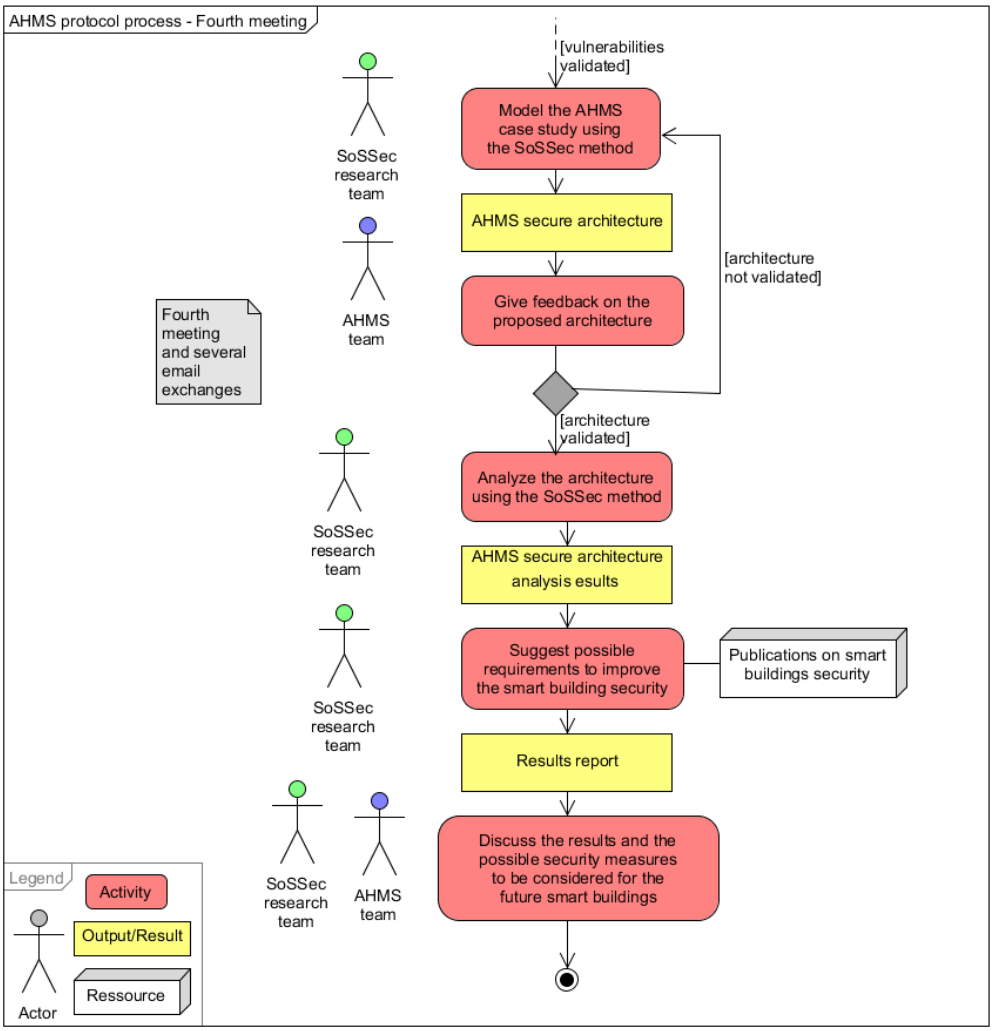


Fig. I.3.4 AHMS planning protocol - Fourth meeting

It is important to mention that the data related to the AHMS components were anonymized for non-disclosure reasons, and all the information related to the real vulnerabilities in the AHMS CSs are not presented for security reasons. Indeed, to define and model the security vulnerabilities and to discover and analyze possible cascading attacks related to the modeled AHMS scenario, we suggested some possible vulnerabilities inspired from our study of the smart buildings security related work including the attack news, which were validated by the AHMS team as realistic.

I.3.4 AHMS smart building modeling

After describing the protocol that we used to conduct the AHMS smart building case study, we present in this section the case study modeling using our SoSSecML DSL and its graphical editor tool and following the SoSSec utilization process we defined.

I.3.4.1 AHMS smart building as a System-of-Systems

With the intention of increasing their functionalities/services and automation, smart buildings are becoming a heterogeneous mix of constituent systems that are complex, autonomous, distributed and highly connected. These CSs are interacting to provide one or many global functionalities not residing in any of these single systems, transforming by that regular buildings into smart buildings that could be engineered as SoS. The Adelaide Health and Medical School (AHMS) is one of these buildings.

AHMS description

AHMS smart building's global goal is to ensure the comfort of its occupants by enhancing the automation to relief them from manual control of different services while increasing the energy savings. This global goal implies several interactions among the independent smart building CSs which are managed by different companies/organizations, and operate independently to achieve their own goals, but also interact to fulfill the AHMS goals.

After several meetings with managers from the AHMS smart building infrastructure branch, we elicited the CSs of the AHMS SoS, the organization that govern each CS and the main operations/functionalities on the CS interfaces employed during the SoS interactions and needed for the achievement of the AHMS global goal, as presented in table I.3.2.

OMGEE specific characteristics in AHMS

Along with the CS interactions, automation, complexity and distribution, we argue that smart buildings, in particular AHMS, are turning into SoS that comply to the **OMGEE** characteristics (defined in the general introduction chapter):

- **Operational Independence of the CSs**: Each CS has its own goals and additionally participates with some functionalities in the realization of the smart building global goal(s). For example, the BMS/HVAC aim is to heat and ventilate the rooms, the smart grid CS objective is to manage the balance between energy production and consumption.

[18]The real names of the organizations were omitted for non-disclosure reasons

Table I.3.2 AHMS smart building description

Constituent System	**Operations on the CS interface**	**Organization**[18]
Building Management System (BMS) or Heating, Ventilation and Air-Conditioning system (HVAC)	Automatically control the heaters, ventilators and air-conditioners through temperature and the exchange of HVAC related information such as temperature and CO_2 levels	Mechanical constructor
Lighting CS	Automatically control the lights according to the occupancy and/or the outside brightness information	Electrical constructor
Smart Grid CS	Manage the energy through energy controlling actions to keep the balance between the energy production and consumption and send energy alerts to the Emergency Management CS in case of power outage	Energy constructor
Audiovisual Management CS	Manage the screen projectors by adjusting the brightness to the needed level based on the type of the projection and the room brightness information	Infrastructure team
Online Scheduling CS	Manage the online booking and broadcast the booking information to the other CSs to adjust the lights, HVAC and audiovisual systems	Appointment scheduling team
Fire Detection CS	Detect automatically the fire in the building through fire alarms and notify the BMS and security surveillance CSs to execute necessary emergency actions	External organization 1
Security Surveillance CS	Manage the access in the smart building, detect intruders and send an intruder alert with the corresponding location to the Emergency Management CS to take the necessary emergency actions	External organization 2
Emergency Management CS	Manage the emergencies in the building based on the emergency alerts and by applying the corresponding emergency actions	Incident management team

- **Managerial Independence of the CSs**: The smart building CSs are managed by different companies/organizations which are responsible of managing the CS components to fulfill their goals, for example the security surveillance CS is managed by an external company to the university of Adelaide and uses its resources (cameras, satellites, database etc.) to detect any intruder in the building;
- **Geographic distribution of constituents**: The CSs in an SoS are loosely coupled in the sens that they do not exchange mass or energy, they only exchange information. This is the case of the smart building CSs that are distributed but connected to communicate and share information;
- **Evolutionary development of the SoS**: Smart building SoS are long-life systems. Their evolution may result from the constituent systems evolution during the Smart building life cycle like the transformation of an existing parking space into a smart parking CS or the introduction of new CSs, like smart waste management and smart water management CSs;
- **Emergent Behaviors in the SoS**: The CSs need to interact to achieve the smart building goal(s) that cannot be accomplished by any of its CSs individually. Even if these interactions could be predictable (for example the management of known smart building emergency scenarios, which have impacts across several CSs: Fire, electricity outage and intruder detection), the specific characteristics of the SoS may lead to unforeseen/unknown emergent behaviors at runtime that affect Smart building functionalities, such as unexpected interactions and security issues.

After arguing that smart buildings in general and the AHMS in particular are Systems-of-Systems with respect to Maier's definition [75], we present in the following sub-sections the detailed SoSSec modeling process and the AHMS secure architecture modeling.

I.3.4.2 AHMS secure architecture modeling using SoSSec

To illustrate the modeling part of our SoSSec method, we used the SoSSecML and its graphical editor to model the structural and behavioral aspects (including security) of the AHMS smart building SoS, following the proposed utilization process (cf. section I.2.6 figure I.2.15).

Define AHMS CS functional and security requirements

As shown in the snippet I.3.5 of the functional and security requirements definition activities, described in our detailed utilization process, we defined the requirements of the AHMS

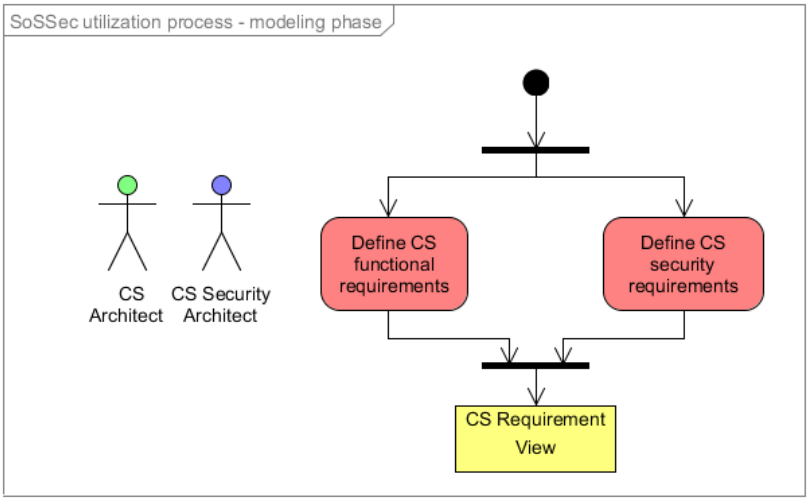


Fig. I.3.5 SoSSec utilization process - Modeling phase - Requirements definition

CSs after analyzing the data collected from the AHMS teams and documentation study as described in the data collection step of our AHMS protocol (cf. section I.3.3). The AHMS architecture team provides us with the functional requirements of each CS in terms of global functionalities/operations that are defined on the CS interface as detailed in table I.3.2, such as the heating and ventilation of the rooms ensured by the BMS/HVAC CS, managing the balance between energy production and consumption achieved by the smart grid CS.

In addition, we used the details of each CS such as the platforms they use and their software components, in order to define a possible list of security requirements for the selected CSs such as detecting the intruders in the smart building ensured by the security surveillance CS and its resources (cameras, satellites, database etc.). Moreover, we conducted a review to study existing publications/documentation on security issues or emergent behaviors targeting smart buildings along with the sources of those issues and behaviors, to define some possible realistic security requirements for our AHMS case study since we cannot present all the real requirements of the AHMS CSs for confidentiality reasons.

Model AHMS CS structural architecture

Having the functional and security requirements of the AHMS CSs, the software and security architect of the SoSSec research team (Ms. EL HACHEM) employed the SoSSecML graphical editor to model the structural architecture of the AHMS CSs as shown in the snippet I.3.6. Using the *Constituent System* concept of our SoSSecML block diagram (cf. SoSSec Metamodel section I.2.3.2 figure I.2.8), the architects separately modeled each AHMS CS presented in table I.3.2 and described for each CS the operations/functionalities

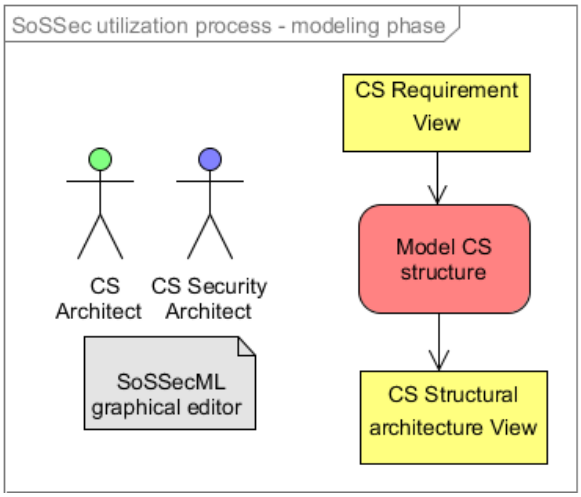
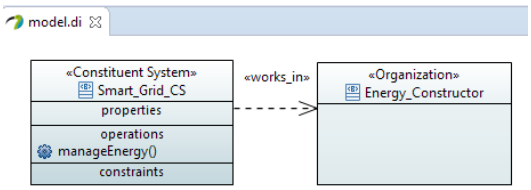


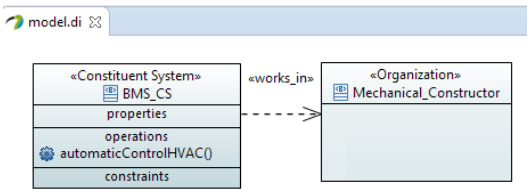
Fig. I.3.6 SoSSec utilization process - Modeling phase - Structure Modeling

on its interface using the block *operation* property. Moreover, the architects represented the company/organization/team that manages each CS using the *Organization* concept and the *works_in* relationship.

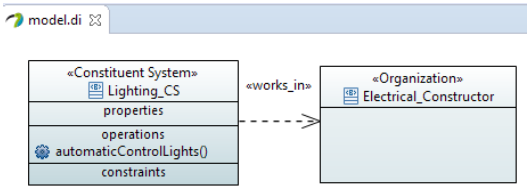
Figure I.3.7 show the resulting structural architecture of the AHMS smart building modeled using SoSSecML and its graphical editor. We can see in figure I.3.7a for example, that the stereotype *Constituent System* of our SoSSecML DSL is used to model the *Smart_Grid_CS*. The *operation* property of this extended block is used to represent the *manageEnergy()* operation, which is one of the *Smart_Grid_CS* global operations/functionalities defined on its interface. The SoSSecML stereotype *Organization* is used to model the *Energy_Constructor* company/organization to which the *Smart_Grid_CS* belong, this association is represented by the *works_in* relationship. All the other AHMS CS are modeled in the same way, each one in a separate block diagram respecting by that the managerial and operational independence characteristics of the AHMS SoS while modeling its structure.



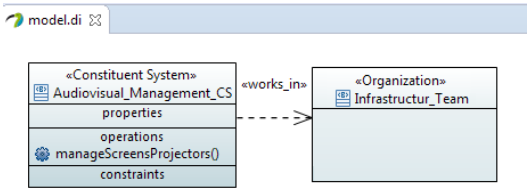
(a) Smart Grid CS block diagram



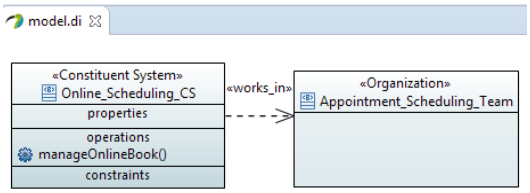
(b) BMS CS block diagram



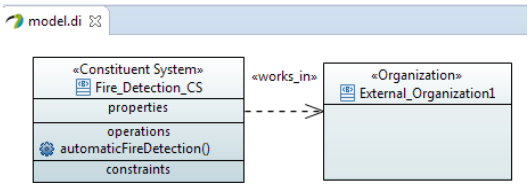
(c) Lighting CS block diagram



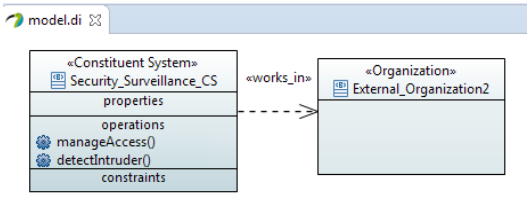
(d) Audiovisual Management CS block diagram



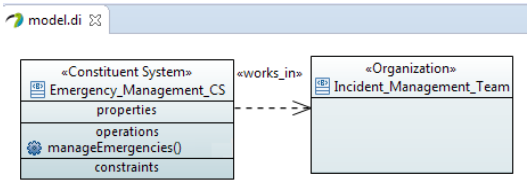
(e) Online Scheduling CS block diagram



(f) Fire Detection CS block diagram



(g) Security Surveillance CS block diagram



(h) Emergency Management CS block diagram

Fig. I.3.7 AHMS CS block diagrams

Model AHMS CS behavioral architecture

Having the functional and security requirements of the AHMS CSs, the software and security architect of the SoSSec research team (Ms. EL HACHEM) employed the SoSSecML graphical editor to model the behavioral architecture of the AHMS CSs as shown in the snippet I.3.8.

Among possible scenarios that represent different interactions between the CSs of the AHMS (table I.3.2), we adopted the following scenario describing some of the SoS interactions during the peek energy consumption hours. In such a scenario, the CSs interact to automatically reduce the energy use while preserving a regular activity and comfort for the occupants:

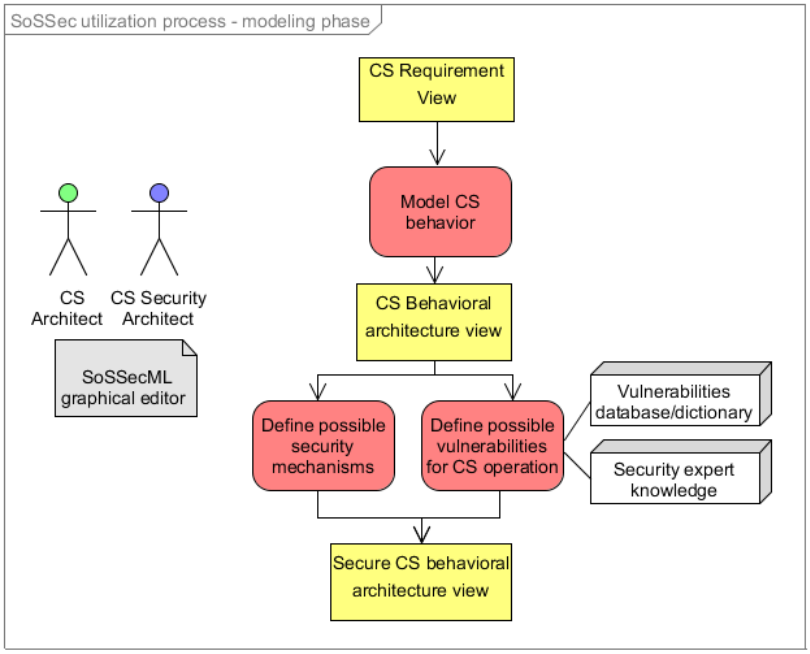


Fig. I.3.8 SoSSec utilization process - Modeling phase - Behavior Modeling

1. To reduce the energy consumption in peak hours, the *SmartGrid CS* interacts with the *BMS CS* via the relation "energy_controlling_actions". Indeed, the *SmartGrid CS* has as one of its functionalities the **manageEnergy** operation to control the energy use of the *BMS CS* through its operation **automaticControlHVAC**;

2. The *BMS CS* has as one of its individual goals/functionalities the automatic regulation of the building ambient temperature based on the information collected from sensors and thermostats. It also uses information from other CSs and sends to other CSs information like the "internal_energy_information (temperature and CO_2 level)" to the *FireDetection CS* for **automaticFireDetection** reasons. In peak hours, the *BMS CS* takes control of the *Lighting CS* to reduce the energy consumption by switching off some lights via the **automaticControlLights** operation and the "light_controling_actions" relationship;

3. In turn, the *Lighting CS* interacts with the *BMS CS* through the operation **automaticControlLights** to send it the "occupency_information" to reduce the energy consumption;

Furthermore, since we cannot present the real vulnerabilities of the AHMS CSs for security reasons, we replaced them by some possible realistic vulnerabilities that we inspired from the Google attack detailed in the first section of this chapter (section I.3.2) and the

studied smart buildings security related work. For each operation in our scenario we defined a list of possible vulnerabilities with their pre and post-conditions.

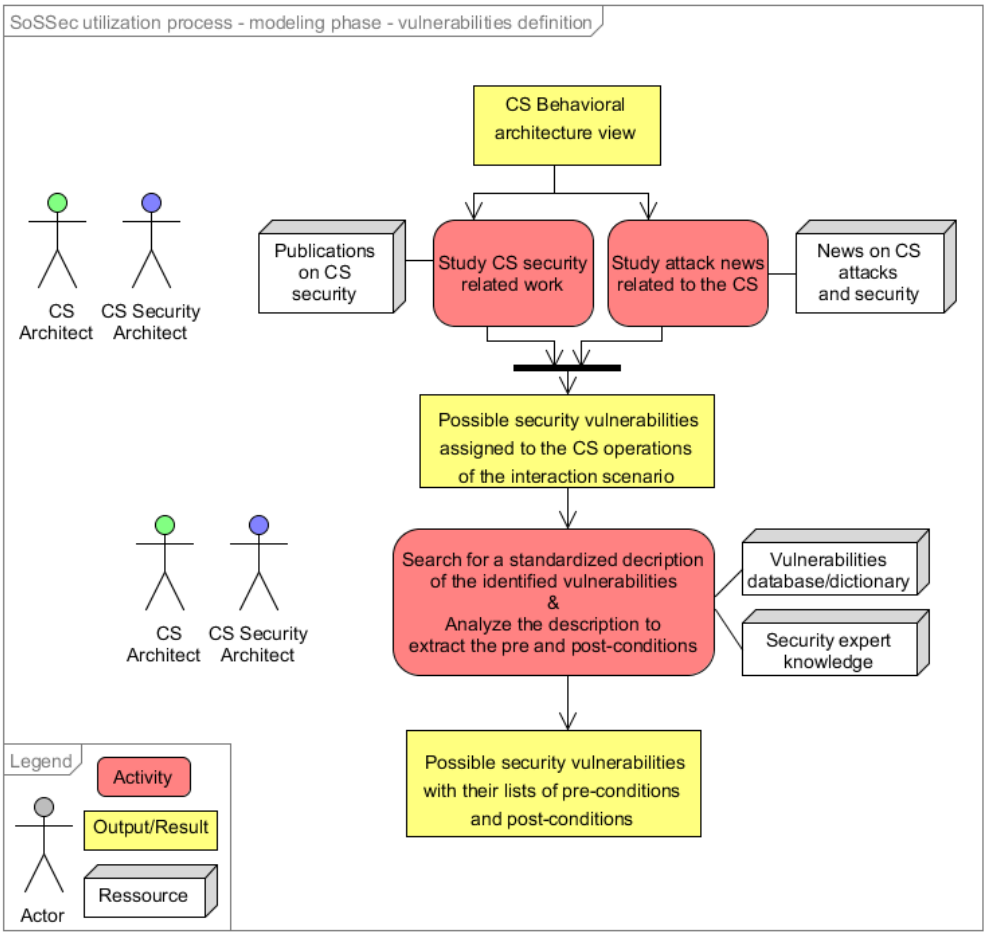


Fig. I.3.9 SoSSec utilization process - modeling phase - vulnerabilities definition

Conforming to the detailed the SoSSec vulnerabilities definition utilization process (figure I.3.9) the software and security architect of the SoSSec research team (Ms. EL HACHEM) studied the existing security publications and attack news to define possible vulnerabilities related to the AHMS CS operations. The architect extracted the description of the vulnerabilities and their condition details from the Common Vulnerabilities and Exposures (CVE) catalog[19]. CVE is a list of common identifiers for publicly known cyber security vulnerabilities, free for public download and use. It is the result of the international cyber security community effort and it is considered as the industry standard for vulnerability and exposure names. For each vulnerability and exposure, CVE offers a standardized description

[19]https://cve.mitre.org/

allowing the extraction of several information such as the vulnerability type and its pre and post-conditions.

Following are the vulnerabilities that we assigned to the different operations of our scenario, with the corresponding lists of pre and post-conditions:

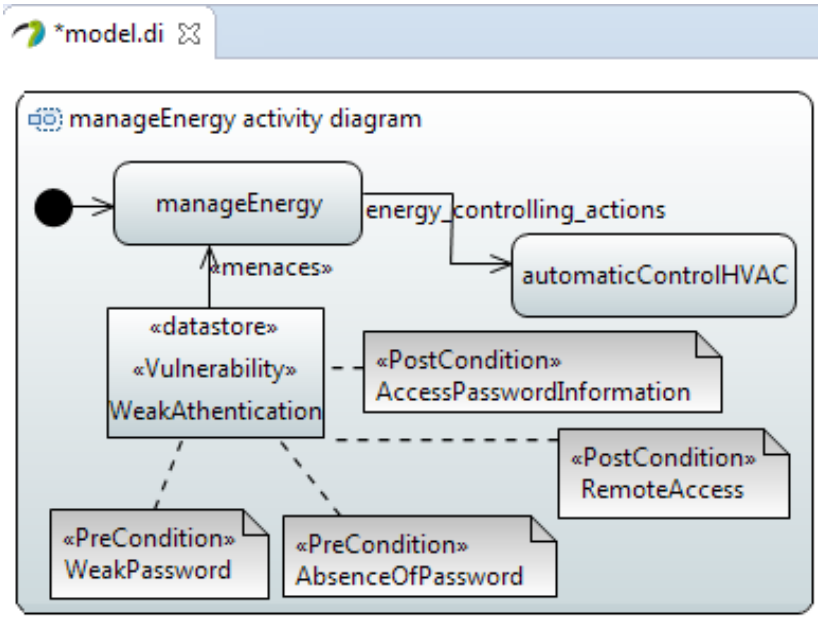


Fig. I.3.10 Smart Grid CS, manageEnergy activity diagram

1. The *SmartGrid* CS contains a large number of wireless sensors that could be open to attacks due to traditional problems like lack of updates and hard-coded/weak/default passwords.Thus, its operations - in particular ManageEnergy (figure I.3.10) - are *menaced* by *WeakAuthentication* vulnerabilities with exploitable *WeakPasswords* or *AbsenceOfPassword* pre-conditions giving rise to the following post-conditions: *RemoteAccess* and *AccessPasswordInformation*. Both post-conditions allow the access to the energy management system of the *Smart Grid* CS and thus to confidential data such as usernames and passwords which could be the same ones used by the *BMS* CS for AutomaticControlHVAC;

2. To the *automaticControlHVAC* operation (figure I.3.11), we assigned two possible vulnerabilities: 1) the *DirectoryTraversal* to bypass access controls, with the pre-condition *AccessPasswordInformation* and the post-condition *InjectArbitraryScript* allowing by that the injection of viruses/warms into the current CS and/or the directly connected CSs such as the Lighting CS; and 2) the *PrivilegeEscalation* vulnerability with one pre-condition: *RemoteAccess* and three post-conditions: *BypassAuthentication*, *DecodePasswords* and *ControlSystem*;

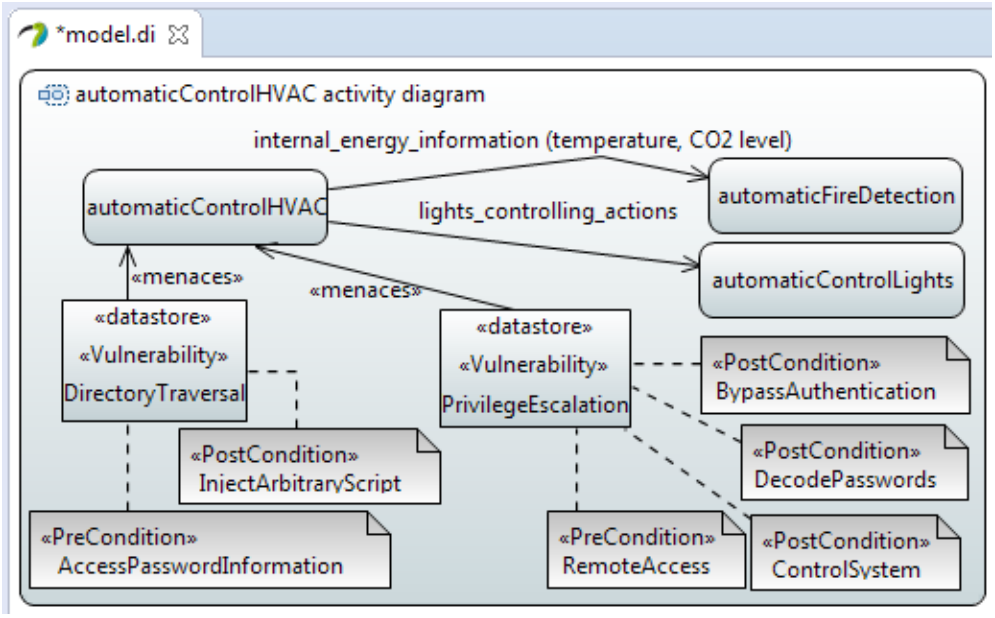


Fig. I.3.11 BMS CS, automaticControlHVAC activity diagram

3. For the *Lighting* CS (figure I.3.12), we can imagine that attackers that have access and control over its operations like *automaticControlLights* through the pre-condition *BypassAuthentication* allowing by that the *InjectArbitraryScript* post-condition. This post condition opens the door to the attackers to create a Mirai botnet for example, enabling Distributed Denial of Services (DDoS) attacks and/or fake emergency alarms.

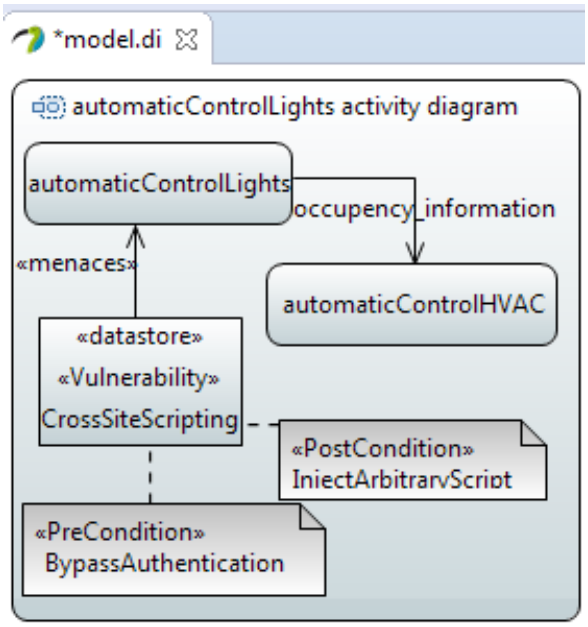


Fig. I.3.12 Lighting CS, automaticControlLights activity diagram

Table I.3.3 synthesizes the assigned vulnerabilities and their description.

According to the SoSSec utilization process (figure I.2.15),the software and security architect of the SoSSec research team modeled the previously described AHMS interaction scenario and the defined security concepts using our SoSSecML and its graphical editor: For each CS, the architect separately modeled the operations that will interact with other operations from other CSs (operations on interfaces) using the *action* concept and the *control flow* relationship of the SoSSecML MetaModel (cf. SoSSec Metamodel section I.2.3.2 figure I.2.8). To the operations, the architect add the corresponding vulnerabilities with the possible pre-conditions that can activate each vulnerability and the list of post-conditions resulting from an activated vulnerability as shown in figures I.3.10, I.3.11 and I.3.12.

As we can see in figure I.3.10 for example, the *manageEnergy* operation defined on the *Smart_Grid_CS* is modeled using the *action* concept of the SoSSecML MetaModel. Its direct interaction with the *automaticControlHVAC* operation defined on the *BMS_CS* interface is modeled using the *energy_controlling_actions ObjectFlow ActivityEdge* of the SoSSec MetaModel. To the *manageEnergy* operation, the SoSSec architect link using the *menaces* relationship the *WeakAuthentication* vulnerability. The latter vulnerability is modeled using the *Vulnerability* stereotype of the SoSSecML MetaModel. In addition, the architects model the pre-conditions *WeakPassword* and *AbscenceOfPassword*, as well as the post-conditions *AccessPasswordInformation* and *RemoteAccess* using respectively the *PreCondition* and *PostCondition* stereotypes of the SoSSecML MetaModel. In the similar way, the software and security architect of the SoSSec research team modeled the two other activity diagrams presented in figure I.3.11 and figure I.3.12.

By modeling separate activity diagrams for each CS, we respect the idea that there is no need to have one SoS architecture team with a global overview on all the SoS interactions and vulnerabilities. In addition the AHMS SoS geographic distribution characteristic was respected by considering that the CSs are physically dispersed and only exchange data (not energy or mass). Moreover, by introducing the security concepts (vulnerability, pre and post conditions) we enrich the behavioral model of the AHMS with meaningful information that will serve, throughout the execution of the model, to detect the possible unknown sequence of activated vulnerabilities (emergent behavior) resulting from the AHMS CS interactions.

Having the AHMS SoS secure architecture represented in one model with different views/diagrams, the next step is to execute this architecture in order to analyze it and detect the emergent unknown security cascading attacks as we will detail in the Part II of this thesis. After running a first analysis of this AHMS architecture and discovering the possible cascading attacks to which the architecture is exposed, the architects may define some security mechanism to avoid possible security issues/attacks identified in the SoS architecture; or they

Table I.3.3 Vulnerabilities related to different AHMS CS operations

CS operation	Vulnerability	Description from CVE	Pre-Condition	Post-Condition
Manage energy	Weak Authentication - CVE-2017-9859 - CVE-2017-9853	"An issue was discovered in SMA Solar Technology products. All inverters have a very weak password policy for the user and installer password... An attacker will likely be able to crack the password... This cracked password can then be used to register at the SMA servers..."	Weak password OR Absence of password	Remote access AND Access password information
Automatic control HVAC	Directory traversal - CVE-2012-4701	"Directory traversal vulnerability in Tridium Niagara allows remote attackers to read sensitive files, and consequently execute arbitrary code, by leveraging (1) valid credentials or (2) the guest feature..."	Access password information	Inject arbitrary script
Automatic control HVAC	Privilege Escalation - CVE-2012-4028 - CVE-2012-3025	"Tridium Niagara Framework does not properly store credential data, which allows context-dependent attackers to bypass intended access restrictions by using the stored information for authentication..."	Remote Access	Bypass authentication AND Decode passwords AND Control system
Automatic control lights	Cross Site Scripting (XSS) - CVE-2014-5382	"Multiple cross-site scripting (XSS) vulnerabilities in Schrack Technik microControl allow remote attackers to inject arbitrary script..."	Bypass authentication	Inject arbitrary script

may just re-configure the SoS architecture to avoid the interactions leading to the cascading attack. Once the new architecture is modeled, the whole utilization process can be followed again for the new SoS architecture iterative to analyze it.

I.3.5 Conclusion

In this chapter, we described the planning protocol of our AHMS case study, we presented the AHMS building as a SoS in respect to the OMGEE characteristics defined by Maier and we analyzed the smart building SoS current security state. Afterwards, we detailed the modeling phase of the SoSSec utilization process to guide the use of the SoSSec graphical editor for modeling the SoS structural and behavioral architecture and the definition of security vulnerabilities. Afterwards, we followed the process to model the AHMS smart building secure architecture.

The modeling phase results in one AHMS secure architecture with several views/diagrams. The modeled architecture illustrates that our SoSSecML DSL and its graphical editor support the modeling of an SoS architecture (in particular the AHMS architecture) while taking into consideration the SoS specific characteristics (precisely the operational and managerial independence of the CSs and their geographical distribution), as well as security aspects to allow the discovery of unknown security emergent behaviors (precisely the possible unknown sequence of activated vulnerabilities).

Part I Conclusion

In this first part, we addressed the secure System-of-Systems (SoS) architecture modeling challenge. We investigated the Model Driven Engineering (MDE) approach to propose a method comprising of a Domain Specific Language: The **S**ystems-**o**f-**S**ystems **Sec**urity **M**odeling **L**anguage (SoSSecML), along with its graphical editor and utilization process, to answer our first research question (RQ1).

We analyzed the existing approaches for secure SoS architecture modeling and we identified their limitations. Our study revealed that: 1) only a very limited number of approaches address secure SoS architecture modeling, and these approach are still at the very early stages and do not present concrete solutions, having been developed in parallel or a little later than our own thesis; 2) there is a need of a new DSL, reasonably based on SysML, to model SoS taking into account their specific properties. The extension should introduce also security concepts to allow the security analysis of SoS architectures, specially that of the cascading attack emergent behavior.

After arguing the lack of approaches that cover our RQ1, we presented the first phase of our SoSSec method to overcome the limitations of existing works. We introduced the SoSSecML abstract and concrete syntaxes which are defined as a profile that extends the SysML syntaxes for secure SoS architecture modeling. One one hand, SoSSecML extends the SysML block diagram to enrich it with concepts specific to the structure of an SoS taking into account its specific characteristics. On the other hand, SoSSecML extends the activity diagram concepts to support the description of the SoS behavior in terms of functional interactions between the CSs and security vulnerabilities. Additionally, we took advantage of the MDE mechanisms to automatically generate a graphical editor tool which extends the Papyrus tool offering by that a strong, portable, customizable and extensible tool that allows the use of our newly introduced concepts besides those of SysML diagrams. Along with the tool, we introduced the process that we followed to build it. This process describes the activities, artifacts and meta-tools used to define the graphical editor. It helps any researcher/engineer to understand the tool building process for further reuse, modifications or extensions.

Afterwards, we illustrated our contribution by modeling a secure SoS architecture of a real-world smart building SoS: The Adelaide Health and Medical School (AHMS). We described the AHMS as a smart building following the SoS specific characteristics and we reviewed the current state of smart building security to identify possible security concerns related to the AHMS CSs. Then we proposed a utilization process that guides the use of the SoSSecML and tools and we followed this utilization process to model the AHMS secure architecture including the block structural diagram and the activity behavioral diagram.

Having secure SoS architecture models and knowing that unknown emergent behaviors mostly arise from CSs interactions in the SoS, the next step is the execution/simulation of these models to analyze these behaviors, in particular the unknown security cascading attacks. We present in Part II the analysis phase of our SoSSec method.

Contribution: The model-driven SoSSecML architecture description language (SysML extension) and its visual graphical editor (Papyrus extension) along with the corresponding processes (graphical editor construction process and SoSSec utilization process for the modeling phase) to model SoS structure and behavior with a particular focus on security aspects (vulnerability, pre-condition, post-condition) that will allow the prediction of unknown security emergent behaviors such as cascading attacks, and its application on a real-world AHMS smart building case study..

Part II

Secure System-of-Systems Architecture Analysis

Part II Introduction

In the first part of this manuscript, we proposed the SoSSec method and detailed its modeling phase to address the secure System-of-Systems (SoS) architecture modeling challenge. In this second part, we present the analysis phase to execute the previously modeled secure SoS architectures and analyze their behavior.

Having SoS models designed at an early stage enables the necessity of analysis techniques to exploit the models' value and approximate the real SoS's behavior in terms of CS interactions and emergent unknown security cascading attacks resulting from these interactions. Indeed, the behavior itself and the related unknown emergent issues become really apparent, and consequently analyzable, only through the execution of the modeled architectures [92].

Model simulation is one of the fundamental techniques to analyze and understand complex system architectures. In SoS engineering, simulation has been identified as a powerful tool to analyze SoS interactions and emergent behavior arising from CSs interactions, as well as to support the early discovery and anticipation of unforeseen issues and failures [92][10][125][129]. Furthermore, the US Department of Defense guidelines for SoS engineering recommend simulation as an effective approach to address SoS complexity and emergent behavior [95]. Nonetheless, only few simulation approaches have been developed or extended to simulate SoS architectures, where the basis of the described models is the SoS behavior or the interactions among the CSs [10].

In this part, we argue the usefulness of Multi-Agent Systems (MAS) for SoS simulation due to the similarities between MAS and SoS concepts. Therefore, the semantics of MAS concepts serves to express the semantics of SoS concepts in an unambiguous way. Moreover, we define a MAS security extension to express the semantics of the SoSSecML MetaModel security-related concepts. In addition, we take advantage of the Model Driven Engineering approach to define a set of transformation rules that map the SoSSecML abstract concepts into executable artifacts inherited from the extended MAS semantics. This alignment allows to efficiently simulate and analyze the secure SoS architectures and discover the possible cascading attacks emergent behaviors, answering by that our second research question (RQ2).

The contributions are organized as follow: In chapter II.1, we review and categorize existing approaches for secure SoS architecture analysis and we identify the most appropriate approach for our needs. In chapter II.2, we propose an extension of a MAS MetaModel to execute secure SoS architectures in order to discover and analyze emergent unknown security cascading attacks. In addition, we follow the MDE approach and use its mechanisms to define a set of Model-To-Text transformation rules to (semi)automatically map the secure SoS architecture to the executable artifacts. Moreover, we complete the SoSSec utilization process with details to guide the use of the method for secure SoS architecture analysis. Afterwards, in chapter II.3, we illustrate our contributions by analyzing the secure architecture of the AHMS real-life smart building SoS, and we discuss the limitations of our case study.

Objective: Propose a Model-Driven simulation method and tools to analyze secure Systems-of-Systems architectures in order to discover unknown security cascading attack emergent behaviors.

Chapter II.1

Related work: Secure SoS Architecture Analysis

II.1.1 Introduction

After presenting the modeling related work in the Part I chapter 1, we now examine the analysis related work. To attain a clearer perception of the secure SoS analysis research current status, we conducted a review of the related existing approaches that have been published up to this date[1].

We followed the same search strategy adopted for the modeling related work. We extracted the publications from the same bibliographic databases (Scopus, IEEE Xplore, ACM digital library, Springer and Wiley online library), in a structured way using several inclusion and exclusion criteria to select relevant studies, however this is not a complete systematic literature review. Moreover, this related work was analyzed having in mind our second research question RQ2, consisting in finding the appropriate analysis technique to execute/simulate the modeled secure SoS architectures and analyze the possible emergent behaviors, instead of only the limitations identified by their own authors.

For that, we added to the set of keywords identified in the modeling related work (Part 1 chapter 1) and related to *SoS*, *security* and *modeling*, some new keywords related to the additional topic of interest: ***analysis***. Indeed, knowing that early discovered vulnerabilities are less expensive to fix than those discovered later, and that unknown emergent behaviors mostly arise from CSs interactions in the SoS, and that these behaviors become apparent and analyzable only when executing the secure SoS architectures, we analyze secure SoS models to discover and avoid emergent security cascading attacks early at the architecture phase of

[1] July 2017

the SoS development life cycle. Therefore, analysis is an essential topic that we included in our review using the keywords "analysis" or "simulation" or "execution".

Table II.1.1 Secure Systems-of-Systems **analysis** approaches

Category	Studied approach	SoS	Security	Modeling	Analysis
D	Nguyen et al., 2017 [90]	✓	✓	✓	✓
	Mori et al., 2017 [84]	✓	✓	✓	✓
	Mohsin et al., 2016 [81]	✓	✓	✓	✓
E	Kobetski et al., 2017 [69]	✓	✓	✗	✓
	Mori et al., 2016 [83]	✓	✓	✗	✓
	Nicklas et al., 2016 [91]	✓	✓	✗	✓
	Guariniello et al., 2014 [47]	✓	✓	✗	✓
	Dahmann et al., 2013 [32]	✓	✓	✗	✓
	Abercrombie et al., 2015 [2]	✓	✓	✗	✓
	Meland et al., 2014 [77]	✓	✓	✗	✓
F	Dickerson et al., 2016 [35]	✓	✗	✗	✓
	Oquendo et al., 2016 [102]	✓	✗	✗	✓
	Harrison, 2016 [56]	✓	✗	✗	✓
	Wachholder et al., 2015 [125]	✓	✗	✗	✓
	Gehlot et al., 2016 [46]	✓	✗	✗	✓
	Alaguvelu, 2016 [3]	✓	✗	✗	✓
	Neto, 2015-2016 [89][88]	✓	✗	✗	✓
	Falkner et al., 2016 [42]	✓	✗	✗	✓
	Guariniello, 2016 [48]	✓	✗	✗	✓
	Zhang et al., 2016 [131]	✓	✗	✗	✓
	Muller et al., 2016 [86]	✓	✗	✗	✓
	Pavon et al., 2011 [103]	✓	✗	✗	✓
	Eusgeld et al., 2011 [40]	✓	✗	✗	✓

We organized the studied approaches in three categories as shown in table II.1.1:

1. Category D: The approaches that jointly address our four topic of interests: SoS AND Security AND Modeling AND Analysis;
2. Category E: The analysis approaches that cover at the same time SoS and security analysis (SoS AND Security AND Analysis);
3. Category F: The analysis approaches that only cover SoS architecture analysis (SoS AND Analysis);

The review presented in table II.1.1 reveals in category D, a very limited number of primary studies, recently published in 2016 and 2017, addressing *Systems-of-Systems modeling* and *analysis* while also considering *security* concerns. We start by presenting these

approaches and the advancements on their realization in section II.1.2. Moreover, since at the beginning of this thesis there were no proposals covering all of our four topics, we further explored the approaches that jointly address *SoS analysis* and *security* related work (category E in table II.1.1) that we present in section II.1.3. This is followed by a deeper review and classification of *SoS analysis* related work (category F in table II.1.1) in section II.1.4. The literature related to the research findings is further discussed in section II.1.5, and the chapter is concluded in section II.1.6.

II.1.2 Existing approaches for secure SoS architecture modeling and analysis

As we can see in table II.1.1, among the reported approaches, only a few novel studies (developed in parallel, in the same time interval, with this thesis) involve conjointly the four previously described topics (category D):

In a recent systematic mapping study [90], Nguyen et al. review model based approaches for security engineering in Cyber-Physical Systems. The study covers many SoS application domains, in particular: Smart grid, smart car, health-care and aircraft systems. The results highlight a noteworthy increase in the number of studies targeting model based software engineering for Cyber-Physical Systems, as well as the effectiveness of Domain Specific Languages (DSLs) and automatic model transformations in this context. To conclude their study, the authors point out to several shortcomings to engineer security for Cyber-Physical Systems, mainly: The lack of solutions, tool support, case studies; as well as the challenge of bridging DSLs and integrating them with security analysis through model transformations for example.

Mori et al. in [84] propose an extension of a visual Architecture Description Language (ADL), precisely a profile to extend the System Modeling Language (SysML) that they develop following the Model Driven Engineering (MDE) process and they use to model a smart grid scenario. They discuss a possible integration of a security viewpoint to the profile to introduce some SoS-independent security concepts such as encryption and access control mechanisms as well as a possible security monitoring infrastructure to detect security incidents and vulnerabilities. The authors mention also the possibility of customizing their tool to generate code for an ulterior analysis using the hazard analysis technique. None of the discussed suggestions is implemented. This work is still at a very early stage, only the profile and a corresponding editor are implemented.

Mohsin et al. in [81] propose a formal framework to analyze the security of the Internet of Things (IoT). They formally specify a general behavioral model for IoT and an interconnected threat propagation tree for IoT using the Satisfiability Modulo Theories. Then, using the proposed framework, they analyze the network configuration against a variety of IoT threats. They claim that their work is the first attempt to model and analyze IoT general behavior and threat resiliency. This study may help analyzing the resilience of an IoT network against specific attack scenarios. However, the IoT is modeled in a generic way with a graph of connected nodes, without considering its specific characteristics as an SoS such as independence and emergent behavior. The analysis is performed based on specific scenarios. Hence this approach addresses a specific application domain, the IoT systems as a complex network, rather then the IoT as an SoS.

After presenting the few related work that jointly cover the four essential topics of our research, we can notice that they are all very recent (2016 and 2017), therefore they have been developed after this thesis was already well advanced in its proposals. Even so, they do not overlap with our proposals, the closest one being [84] in its extension of SysML. However, our approach in addition to being more advanced in the implementation and application of its proposals, also uses the Multi-Agent Systems for secure SoS architecture analysis.

II.1.3 Existing approaches for secure SoS architecture analysis

In this section, we review the secure SoS analysis approaches. We analyze the presented approaches regarding their ability to analyze SoS taking into account their specific characteristics as well as their ability to achieve any kind of security analysis. Subsequently, we examine the strength 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 tried to regroup them in the following sub-categories: 1) Approaches based on threat analysis processes, 2) Graph-based approaches and 3) Goal-oriented/Agent-based approaches.

II.1.3.1 Approaches based on threat analysis processes

Kobetski et al. investigate in [69] 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. [83] 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.

II.1.3.2 Graph-based approaches

Nicklas et al. in [91] 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 [47] 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 [47], Dahmann et al. [32] 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. It is more a performance evaluation than a security analysis.

II.1.3.3 Goal-oriented/Agent-based approaches

Abercrombie et al. in [2] 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 [77] 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.

II.1.3.4 Discussion

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

1. The approaches based on threat analysis processes [83][69] only focus on static scenarios to analyze the distribution of threats and hazardous states. They lack of 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 [91][47] [32] 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 [2] [77] 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, which limits the benefits of the SoS security analysis to the ability or probability of maintaining SoS operations when an attack occurs. 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 (defined in the introduction chapter) in the analysis. Moreover, they lack efficient 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 issues.

For all these reasons and with the intention of extending the SoS analysis related work, we investigate in the next section the approaches targeting the analysis of SoS without considering security properties (*SoS* AND *Analysis* - category F in table II.1.1).

II.1.4 Existing approaches for SoS architecture analysis

In recent years, diverse approaches were used to execute and analyze SoS architectures (category F in table II.1.1). We categorized them into six sub-categories as follows:

1. *UML-executable based approaches*, for example foundational UML (fUML[2]) and Action Language for Foundational UML (ALF[3]). These languages are an intermediary between UML models and platform executable languages. They describe executable semantics, specified in first order logic and based on Process Specification Language, for UML-based models;

[2]Foundational Subset For Executable UML, http://www.omg.org/spec/FUML/1.1/
[3]Action Language For Foundational UML, http://www.omg.org/spec/ALF/

2. *Graph models or probabilistic graph models* such as Bayesian networks. They use graphs to express the conditional dependencies structure between random variables;
3. *Colored Petri nets* that provide formal foundation to analyze concurrent systems;
4. *Bio-inspired approaches*, such as living cells and genetic algorithms. They investigate the similarities between human body and complex SoS, for example the organs/tissues/cells represent CSs;
5. *Event-based approaches* such as Discrete Event System Specification (DEVS). They provide a formalism to analyze general systems that can be described as events/states and transitions;
6. *Agent-based approaches* which are computational models formed by a combination of artificial intelligence and software engineering. They represent new means of analyzing complex software systems by simulating the actions and interactions of autonomous agents with a view to assessing their effects on the system as a whole.

We present in the following sub-sections the main existing work belonging to the described sub-categories and we discuss their efficiency in considering the SoS characteristics and CS interactions.

II.1.4.1 UML-executable based approaches

Dickerson et al. [35] also suggest the use of fUML and ALF to associate formal semantics to SysML models for traffic management SoS. The statements of the SysML languages are interpreted to mathematical matrix representation with the intention to apply one of the traditional formal methods such as model checking, formal testing, and proving of theorems. This approach could be efficient to formalize the behavioral concepts, however it remains specific to one SoS application domain. Moreover, the approach in its current form does not handle the emergent behavior, neither express the specific SoS characteristics.

Oquendo et al. [102] propose an executable SysADL language proposed in [101]. The executable notations are based on the action language ALF and fUML. Three executable elements are discussed in this work: The executable construct to specify the body of actions, along with its parameters for data exchange and its body to define how the exchanged data is computed. By adding this viewpoint, the authors offer a language to specify and execute structural and behavioral aspects for SoS. However, they do not argue the selection of ALF and fUML to properly express the semantics behind SoS concepts, neither do they clarify how the proposed executable language could handle the SoS characteristics, mainly the emergent behavior.

II.1.4.2 Graph models or probabilistic graph models

In [56] Harrison explores the graph theory and known graph algorithms to analyze some aspects of the SoS such as complexity of deployment and risk of failure. The proposed approach underline the need of well defined interactions between CSs to understand the overall SoS complexity thus the use of several complexity metrics inspired from different domains (such as information theory and biology). The work is at its initial stages and the author concluded with the need of extending the proposed algorithm and the need of techniques to simulate and analyze SoS and the interfaces between CSs.

In [125] Wachholder et al. propose a bigraph-based approach to analyze the SoS emergent behavior. The bigraphs allow representing the static structures of physical spaces based on the concepts of agent, placing, and linking. The emergent behavior is formally specified in terms of context-sensitive interactions and reaction rules, limiting by that the analysis capacity of the approach to known emergent behaviors.

II.1.4.3 Colored Petri nets

Gehlot et al. introduce in [46] an approach to simulate large scale systems using colored Petri nets which are an extension of Petri nets to model and simulate software systems. The approach allows the specification of places, transitions and arcs to connect the places and transitions expressing the causes and their effects. The advantage of the described approach resides in the verification and validation of the models. However, the approach lacks elements to express SoS semantics of its specific characteristics specially those of the emergent behavior.

II.1.4.4 Bio-inspired approaches

In [3] Alaguvelo et al. propose a genetic-based approach to analyze the architecture of a transportation SoS and search for the optimal SoS architecture from a population of randomly initialized SoS architectures based on a set of performance parameters such as smartness, safety and efficiency expressed using an "SoS fitness value". While the approach seems promising to evaluate the performance of an SoS architecture, however, the parameters used are specific to the transportation domain and the emergent behavior is not considered.

Neto et al. in [89] suggest biological inspired approaches for SoS simulation. They investigate the use of living cell software-based simulation to study the SoS complex behavior. The authors argue the use of living cells by comparing the SoS characteristics and a cell (as an alive SoS). Nonetheless, they do not detail what cell simulation techniques could be used,

neither do they clarify how the cell approach could be adapted to simulate SoS, specially to handle the managerial and operational independence of CSs.

II.1.4.5 Event-based approaches

In a more recent work, Neto [88] introduces a MDE approach aiming to validate the emergent behavior of SoS. The author defines a set of transformation rules to transform the concepts of the SosADL language proposed by [100] into DEVS elements for simulation. In this preliminary work the author does not argue the choice of DEVS for SoS simulation, specially from a semantic point of view. Only the emergent behavior is considered in the mapping, while the other SoS specific characteristic are not discussed.

Falkner et al. [42] propose a framework for analyzing SoS performance. The approach is based on the System Execution Modeling which they suggest to deploy on a hardware testbed to be simulated and to use scenarios for analyzing the system performance under various constraints. The suggested framework seems to be promising since it considers the use of MDE, thus it could be extended to consider other SoS non-functional properties. However, the authors do not precise what techniques could be used for the simulation and analysis, neither how to use them. In addition, it is not clear how the proposed approach handles the SoS specific characteristics.

II.1.4.6 Agent-based approaches

Guariniello et al. introduce in [48] a Functional Dependency Network Analysis approach to examine the operational dependencies between SoS components by giving insights into the causes of the observed behavior. They validated their approach through agent-based simulations. Another approach based on the Functional Dependency Network Analysis is proposed by Zhang in [131] to analyze the safety of the Global Navigation Satellite SoS. While the proposed approaches capture respectively the functional dependencies between the CSs and the hazard effects of such interactions, they do not cover the SoS behavior details, neither the unknown emergent behavior nor do they discuss the other SoS specific characteristics. Moreover, the analysis is not completely objective since it is based on some user-defined parameters and assumptions such as criticality and self-effectiveness.

Muller et al. in [86] propose an agent-based approach to simulate counter-traffic SoS. The authors concluded that agent-based simulation improves the evaluation of the SoS non-functional properties, in particular performance. The authors in this work focus on one specific SoS domain when specifying the simulation environment. Only the known functional behavior is considered in the SoS.

Pavon et al. propose in [103] an agent-based simulation approach to analyze and validate SoS architecture by checking that the specifications meet the requirements and the consistency of those specifications. The authors recommend simulation as a useful tool to understand SoS dynamics. This approach seems interesting for the execution of SoS models, however it does not propose a thorough solution since SoS characteristics and semantics are not considered when mapping from the specifications to the simulation.

Eusgeld et al. propose in [40] the creation of "what-if" SoS alternative scenarios using the High Level Architecture standard, and the simulation of these scenarios to analyze the interdependencies between industrial control CSs, in particular the CSs of the Supervisory Control and Data Acquisition SoS. The evaluation of the approach testify the usefulness of agent-based simulation to analyze SoS interdependencies. The authors mention that their platform is limited when dealing with geographic and logic interdependencies which limits its effectiveness for the analysis of all SoS aspects.

II.1.4.7 Discussion

After analyzing the existing SoS analysis studies (category F in table II.1.1) belonging to the previously identified sub-categories, we summarize below the advantages and limitations of each sub-category:

1. UML-executable based approaches allow the the application of existing formal methods such as model checking and formal testing. However, they do not address the unknown emergent behavior resulting from the SoS interactions;

2. Graph models analyze the operational interdependencies and study the impact of attacks on the SoS operability. Nonetheless, they use subjective parameters specific to the studied application domain for the analysis, without information about the root cause of the analyzed behavior/effect. In addition, they suffer from several shortcomings when dealing with large-scale complex SoS;

3. Colored Petri nets help identifying the cause-effect relations mainly beneficial for model verification. Yet, they lack elements to express SoS semantics and do not address the emergent behavior;

4. Bio-inspired approaches allow the identification of hazards. However, these approaches are in their early stages and suffer from limitations when addressing the CSs operational and managerial independence, thus they need further exploration to be applied in the SoS context;

5. Event-based approaches consider the emergent behavior in the analysis. Even so, they are limited when addressing sequences of events and the reasons leading to unknown emergent behaviors;

6. Agent-based approaches simulate the interactions (actions and re-actions) in the SoS and reveal detailed information about the simulated elements in the simulation results. Although, with such approaches there is a need for well defined specifications of the SoS architecture to reveal the proper SoS semantics and improve the simulation results.

II.1.5 Related work discussion

Our review shows that several approaches have been recently proposed to address the SoS analysis challenge. Many of them could be promising, yet none of the studied works fully address our main objective, that of *modeling* and *analyzing secure SoS* architectures to discover the security cascading attacks emergent behavior, at the architectural stage of the SoS development life cycle.

The only related work that approximatively addresses our objective is that of Mori et al. [84]. Nevertheless, the proposed approach does not cover all the SoS characteristics when modeling the structure and behavior of the SoS architecture: It only describes the CSs and their interactions using an extension of the block diagram. It does not handle the managerial and operational independence of these CSs, neither the unknown emergent behavior, it only describes the known possible emergent behaviors using the sequence diagram. The authors mentioned the possibilities of generating code to realize hazard analysis, but this idea is not detailed and there is no description of any mapping rule to generate the code to a specific platform without loosing the semantics of the SoS architectures, nor any argumentation of the choice of the hazard technique for the analysis.

II.1.5.1 Static vs. dynamic analysis approaches

Given their diversity, the studied approaches could be regrouped with respect to two views: 1) **static formal approaches** assembling UML-executable based approaches; graphic models or probabilistic graphic models; Colored Petri nets; and 2) **dynamic simulation approaches** including Bio-inspired approaches, Discrete EVent System Specification (DEVS), Agent-based approaches.

Each of the two views reflects a different aspect of SoS architecture analysis. The static formal approaches mainly suffer from the combinatorial explosion issue, resulting from the analysis of all the possible interactions and attacks in the SoS. Not to mention that formal

approaches use complex mathematical formalisms to model and analyze the SoS architecture, therefore usually the proposed solutions encounters complexity problems, specially from the user/industry engineers point of view.

On the other hand, the dynamic analysis approaches seem to be well suited for SoS execution and analysis mainly to handle the emergent behaviors resulting from CS interactions, as well as to support the early discovery and anticipation of unforeseen issues and failures that need to be corrected in the specification level before the integration step can be conducted [92][10][125][129]. Dynamic approaches implements the fundamental dynamics associated to the SoS interactions, and they help identifying the key factors that influence the overall SoS behavior. Several studies related to system engineering conclude that dynamic simulation is a fundamental technique to analyze and understand complex system architectures [117][111]. Furthermore, the US Department of Defense guidelines for SoS engineering recommend simulation as an effective approach to address SoS complexity and emergent behavior [95].

II.1.5.2 Multi-Agent Systems simulation

Moreover, our related work results show that agent-based simulation seems the most suited and actively used technique to express SoS aspects (at least by the number of agent-based SoS analysis related work which is relatively high when compared with other approaches). Agent-based simulation approaches were extensively used with the intention to execute the architectural choices and understand the impact of CS behavior on the SoS behavior.

Agent-based approaches have several advantages when exploring the SoS behavior [10]. By performing agent-based simulation, the SoS behavior could be understood enabling the SoS architects to improve the SoS design structure and behavior (functionality) while maintaining control, minimizing vulnerability, and increasing agility of the SoS.

Indeed, Multi-Agent Systems (MAS) approaches 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.

Moreover, 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, pro-active 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;
- Self steering: Agents in MAS maintain a chosen course without constant human action;
- Large problem solving: MAS intend to solve realistic and large scale problems that are beyond the capabilities of an individual agent;
- Legacy systems leveraging: No need of rewriting of the system, MAS allow the incorporation of new agents by only writing the agent wrappers.

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 the agents environment.

II.1.6 Conclusion

Our related work study of the existing analysis approaches for secure SoS architectures shows the limitations of the existing approaches. Moreover, the study shows that a Multi-Agent System (MAS) simulation method is a solid base for SoS architecture analysis. However, only a few of the studied MAS simulation approaches for SoS analysis existed at the beginning of this thesis ([103][40]), most of them were developed afterwards, in parallel with our work. They mostly propose guidance or suggestions or early stages ideas without offering efficient tools or deep security analysis. None of them offers proper execution artifacts to execute secure SoS architectures and analyze them to discover security emergent behaviors, thus none of the existing approaches fully addresses our second research question RQ2.

Using MAS in our method should nonetheless introduce well-defined relations between the secure SoS models developed using our SoSSecML language, and their realizations/execution. The simulation/executable artifacts should express the semantics of the DSL/modeling approach artifacts, to accurately capture the real behavior of the SoS.

In addition, the approach should be extended to include security executable artifacts, in order to express the SoSSecML security concepts such as vulnerabilities, pre- and post-conditions and attacks. These concepts could be inspired from the existing approaches presented in the first section of first chapter of Part I, and analyzing SoS security and failures. Nonetheless, the security analysis techniques should be improved to avoid combinatorial explosion and discover effectively the sequence of unknown vulnerabilities triggered only due to the SoS interactions and security breaches, instead of analyzing all the possible interactions and attack paths.

In the next chapter, we argue in details the usefulness of Multi-Agent Systems (MAS) for SoS agent-based simulation due to the similarities between MAS and SoS concepts. Moreover, we extend MAS with security concepts to express the semantics of all the SoS architecture artifacts/concepts defined by our SoSSecML language, offering by that an efficient and unambiguous agent-based simulation approach for secure SoS architecture.

Chapter II.2

SoSSec: A Model driven method for secure SoS architecture analysis

II.2.1 Introduction

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. In the context of secure SoS, as we began arguing in the previous chapter (cf. subsection II.1.5.2), Multi-Agent Systems (MAS) simulation is a promising analysis technique that could help designers and engineers to understand the interaction between the independent constituent systems of the SoS, and to analyze the SoS as a whole.

In this chapter, we present the analysis phase of our SoSSec method. We propose an extension of a MAS MetaModel to study the secure SoS behavior, analyze the conditions and the ways in which a CS could be vulnerable to an attack, and to predict the possible sequence of triggered vulnerabilities connected through the interactions between the CSs of the SoS to form high-impact cascading attacks affecting the entire SoS, answering by that our second research question RQ2.

Given that a method is defined by a set of modeling conventions (the modeling language) and a process to provide guidance on the use of the language, along with the corresponding actors and required tools, the contributions in this chapter are organized as follows: In section II.2.2 we introduce the basics of agents and MAS. We then explain, in section II.2.3, the SoSSecML semantics inherited from MAS by detailing the similarities between the SoS and MAS domains. In section II.2.4 we present the MAS security current state, and given the lack of MAS security concepts that express the SoSSecML security concept semantics, we define our security extension of the MAS MetaModel. Then we compare a collection

of MAS platforms and we argue the selection of the JADE platform that we extended to implement the extended MAS MetaModel. In section II.2.5, we detail the set of Model-To-Text (MTT) transformation rules - the SoSSec generator tool - that we define to generate MAS executable/simulation artifacts and thus implement the defined alignments between the SoSSecML and extended MAS MetaModel concepts. In section II.2.6 we describe the secure SoS architecture simulation and the log files creation. Section II.2.7 presents the analysis phase of our SoSSec utilization process and we conclude the chapter in section II.2.8.

II.2.2 Multi-Agent Systems (MAS) basic concepts and standardization

The agent technology started in the distributed artificial intelligence domain. Later on, autonomous agents and Multi-Agent Systems (MAS) were explored to analyze and implement complex, dynamic and distributed software systems where unplanned behavior is to be expected [93]. In the following subsections we define agents, MAS and the Foundation for Intelligent Physical Agents specifications (FIPA) standardization.

II.2.2.1 Agents

An agent is any autonomous entity that plays certain roles and interacts with other agents to achieve individual or common goals. It is essentially a software-based computer system that enjoys the following properties: *autonomy, social ability, re-activity* and *pro-activity* [126]. In addition, some authors consider several other properties, such as *mobility, veracity, rationality* and *learning* [112]. Table II.2.1 details these properties.

II.2.2.2 Multi-Agent Systems

MAS are distributed software systems composed of multiple autonomous/independent agents that interact in a specific environment to accomplish local individual as well as common/collaborative goals. Figure II.2.1 present a typical view of a MAS with multiple agents perceiving and acting with their environment and interacting with each other through communication. MAS are decentralized systems without global control. Each agent in MAS has incomplete information or capabilities for solving the problem, therefore, each agent has a limited viewpoint. Table II.2.2 outlines the main advantages of MAS collected from [126][93][112].

Table II.2.1 Agent properties

Agent property	**Description**
Autonomy	Agents are at least partially independent and self-aware. They operate and control their action without the direct interventions of humans or others
Social ability	Agents interact through agent-communication languages
Re-activity	Agents perceive their environment and respond to changes
Pro-activity	Agents are able to perform direct behaviors by taking the initiative
Mobility	Agents have the ability to move around their environment
Veracity	Agents do not knowingly communicate wrong information
Rationality	Agents act in order to accomplish their goals to the extent that their beliefs permit
Learning	Agents adapt their behavior to improve their performance

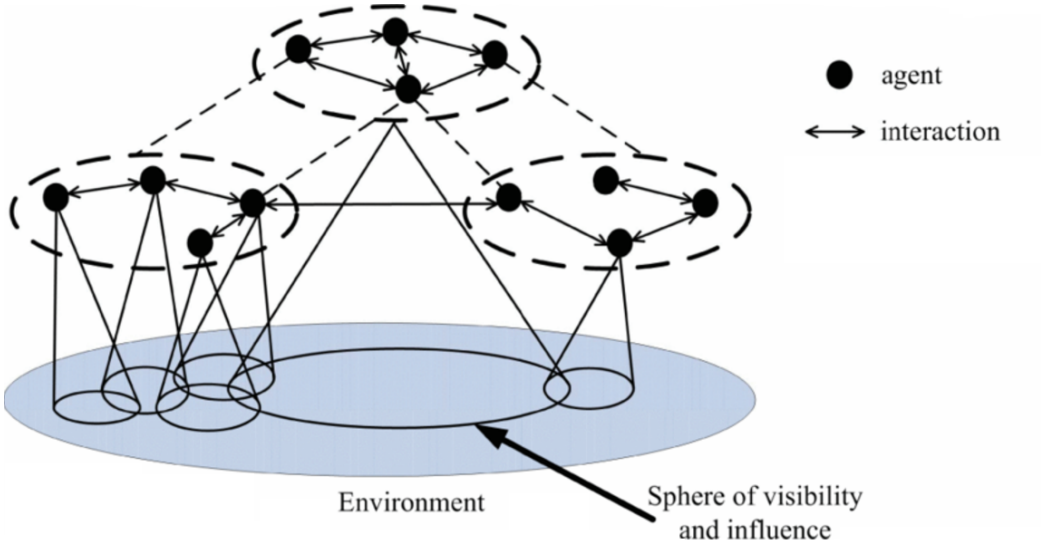


Fig. II.2.1 The typical structure of a Multi-Agent System from [93]

II.2.2.3 MAS specifications: The Foundation for Intelligent Agents (FIPA) standards

Considering MAS advantages (cf. table II.2.2) and their wide usage in the academic community, and with the intention of promoting their usage beyond the exploratory perspective, the Foundation for Intelligent Agents (FIPA) established a consortium composed of academic and industrial organizations, to develop a core set of specifications for MAS called "FIPA specifications" and available for download from *http://www.fipa.org*.

Table II.2.2 MAS main characteristics and advantages

MAS advantages	Description
Computational efficiency	Through asynchronism and concurrency
Reliability	Ensured by redundancy, through a backup pool of agents, beyond the necessary number required to act on a task, that have experience on that task
Extensibility	The number of agents in a MAS and their capabilities could be extensible
Maintainability	Via modularity which consists in structuring an individual agent's program in separate modules, each module serving a specific functionality and can be used to handle specific situations or tasks
Reuse	Agents can be reused in different groups/organizations to solve different problems

The fundamental abstract and implementation concepts underpinning our use of MAS are conform to the following FIPA specifications[1]:

- FIPA-Specification SC00001[2], FIPA Abstract Architecture Specification: Defines abstract specifications for MAS - specifications grounded in particular technologies, representations, and programming models - such as agent, agent attribute, agent behaviors, Agent Communication Language (ACL), content language, message elements;
- FIPA61-Specification SC00061[3], FIPA ACL Message Structure Specification: Defines specifications for the FIPA ACL message parameters, to standardize the form of a FIPA-compliant ACL message, in order to ensure interoperability by providing a standard set of ACL message structure;
- FIPA8-Specification SC00008[4], FIPA SL Content Language Specification: Defines a concrete syntax for the FIPA Semantic Language (SL) content language. This syntax and its associated semantics are suggested for use in conjunction with the FIPA ACL. A FIPA-SL content expression may be used as a content of the ACL message. The expression could be a proposition (which may be assigned a truth value in a given context), an action (which can be performed), an identifying reference

[1] http://www.fipa.org/repository/standardspecs.html
[2] http://www.fipa.org/specs/fipa00001/index.html
[3] http://www.fipa.org/specs/fipa00061/SC00061G.pdf
[4] http://www.fipa.org/specs/fipa00008/SC00008I.pdf

expression (which identifies an object in the domain) or a composition of these three basic expressions.

In our contribution, we will use the above MAS abstractions/concepts to express the semantics of the SoSSecML syntactic concepts, by aligning each of the SoSSecML MetaModel concepts with an appropriate MAS MetaModel concept which articulate similar/analogous meaning, as detailed in the next section.

II.2.3 SoSSecML semantics: Alignment of concepts between SoS and MAS MetaModels

To perform useful analyses of the SoS emergent behavior arising from CS interactions, our secure SoS modeling and analysis method (SoSSec) should provide a solid alignment between the modeling domain and the simulation domain where the models will be executed and analyzed. Indeed, the analysis of the architectures conceived using a DSL should underly the semantics of the model and its specification features [76]. The execution (executable artifacts) of the defined architectures should hold the properties described in the models [1]. Consequently, having developed the abstract and concrete syntaxes of our SoSSecML in the first part of this thesis (sections I.2.3 and I.2.4) to allow the modeling of secure SoS architectures, we now define SoSSecML semantics that will allow the execution and analysis of the secure SoS architecture, following the MDE DSL definition approach, presented in subsection I.2.2.1.

Consequently, a first step in our analysis phase is to define a consistent alignment between the SoSSecML MetaModel (modeling domain) and the MAS MetaModel (simulation/execution domain). This will permit, in a second step, a machine assisted (in our case a MAS platform) execution of the secure SoS architectures to register the real structure and behavior of the described SoS, including the cascading attack emergent behavior.

Therefore, we now present the MAS MetaModel concepts that we will use to properly express and interpret in an unambiguous way the semantics of the secure SoS architectures conceived conforming to our SoSSecML.

II.2.3.1 MAS MetaModel

Based on the MAS specifications defined by the FIPA (cf. subsection II.2.2.3, we represent in figure II.2.2 a part of the MAS MetaModel that we designed using Ecore[5]. We will use the

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

MAS concepts presented in the figure, mainly *Agent, Behavior* and *ACLMessage*, to express the SoSSecML semantics.

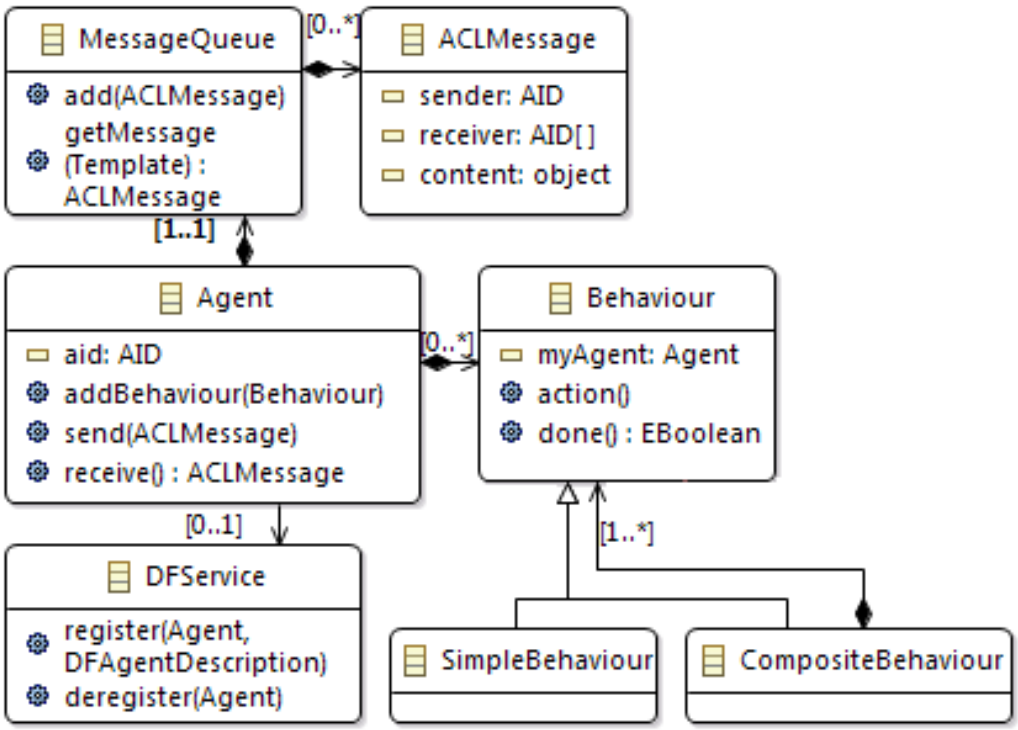


Fig. II.2.2 Part of the MAS MetaModel

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 stoked.

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).

II.2.3.2 SoSSecML MetaModel and MAS MetaModel concept alignments

Having in mind the MetaModel (syntactical concepts) of our SoSSecML DSL (defined in Part I I.2.3.2), we aligned them with the corresponding MAS MetaModel concepts based on the similarities between them allowing by that an adequate alignment between both SoS and MAS domains. Table II.2.3 exhibits the essential similarities/alignments between the SoSSecML MetaModel concepts and the MAS MetaModel concepts. In what follows, we argue and explain these similarities.

Table II.2.3 Essential similarities between SoSSecML MetaModel and MAS MetaModel concepts

SoSSecML 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 (by working_in/belonging to different organizations).
Operation/Action	Behavior/Task	Simple-complex operations/simple-composite behaviors; sequential and parallel/concurrent operations/behaviors.
Object Flow	ACL Messages	Express interactions/communications between the operations/behaviors defined on CS/Agent interfaces.

Constituent System and Agent

This first alignment is between the *Constituent System* (CS) concept of the SoSSecML MetaModel and the *Agent* concept of the Multi-Agent Systems MetaModel.

As described in the first part section I.2.3.2, 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 dimensions/organizations that can be distributed over the network (managerial independence).

For all these similarities, the *agent* concept in MAS can properly express the semantics of the *CS* concept of SoSSecML.

Operation/action and Behavior

Another correspondence is between the *operation/action* concept of the SoSSecML MetaModel (cf. section I.2.3.2) and the *behavior/task* concept of the MAS MetaModel.

With SoSSecML, a CS operation modeled using the *operation* property of the CS block structural concept, and represented also in the activity diagram using the *action* concept, describes the CS global functionality, the operation on its 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.

Based on these similarities, the agent *behavior* concept of the MAS MetaModel can adequately express the semantics of the CS *operation/action* concept of SoSSecML MetaModel.

Object Flow and ACL Messages

Another similarity is between the CS interactions in SoS and agents interactions in MAS; consequently between the *Object flow* concept of the SoSSecML MetaModel (cf. section I.2.3.2) and the *ACL message* concept of the MAS MetaModel.

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 modeled using the *ObjectFlow* notation of the activity diagram of our SoSSecML DSL.

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, such as FIPA-SL (previously described in subsection II.2.2.3), and content expressions can be grounded by referenced ontologies.

For these resemblances in the meaning they express, the agent *communication/ACL message* concept in MAS can appropriately express the semantics of the CS *interactions/Object flow* concept of SoSSecML.

II.2.3.3 Discussion

As our analysis shows, MAS seem to be an appropriate choice for SoS simulation. Although they lack concepts to express SoS detailed structural and security aspects, MAS could be a good complement for DSLs to execute/simulate rich SoS models - including structural and behavioral properties described with our SoSSecML.

However, to express the dynamics and execute the whole behavior of the modeled 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 perform a complete alignment between SoSSecML and MAS semantics.

Therefore, it is necessary to assign executable MAS notations to the SoSSecML security notations, mainly the vulnerability, pre-condition and post-condition concepts presented in our MetaModel (cf. I.2.3.2). Accordingly, we explore in the next section the security realizations in MAS.

II.2.4 SoSSecML semantics: MAS MetaModel security extension

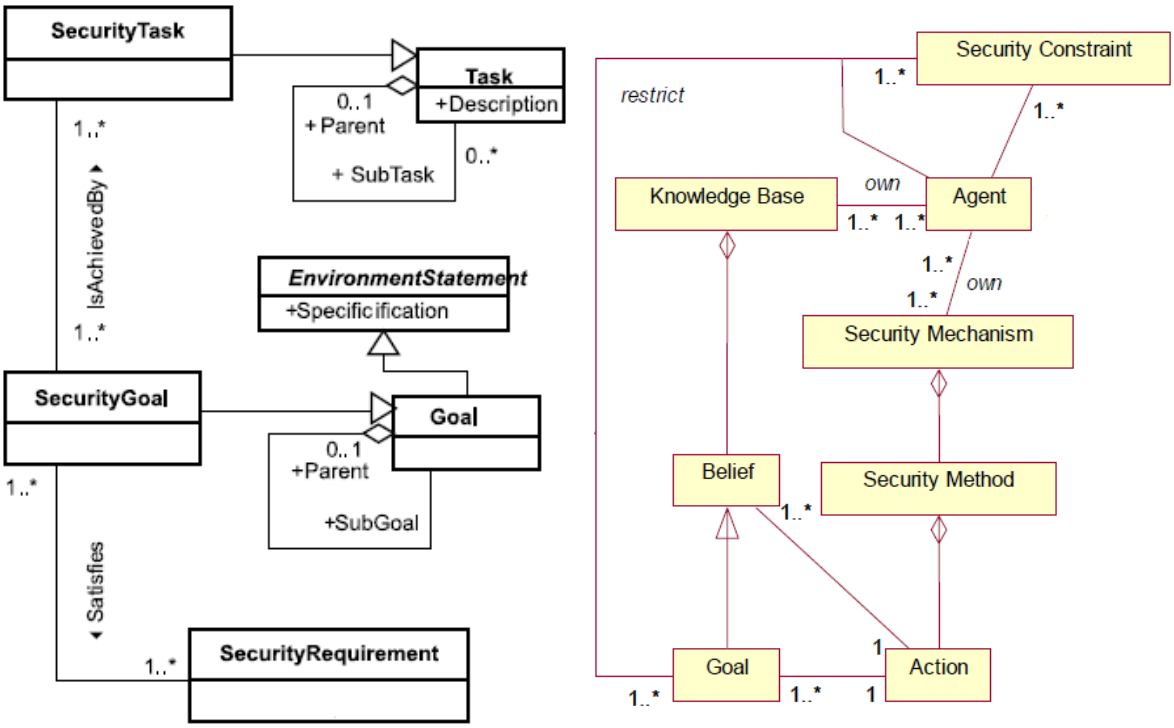
There is a growing interest in using MAS for complex systems simulation and analysis and increasing number of agent projects, platforms and applications conform to the FIPA standard. Still, the existing FIPA standards[6] do not deal with security. There were only a few limited attempts to add security to FIPA, mainly the key management and authentication [107], however, the document did not mention how the security should be specified. FIPA recognized the lack of specifications to deal with security and issued a call - today outdated - for work in the area [114].

[6] http://www.fipa.org

In this section we study the current state of security in MAS and we propose and implement an extension of MAS MetaModel for cascading attacks detection.

II.2.4.1 Current security state in MAS

In the past years many attempts have been made to include security into MAS. However, the following recent surveys [57][96][16][23] 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.



(a) Part of the MetaModel from [14]

(b) Part of the MetaModel from [13]

Fig. II.2.3 Security-aware MAS MetaModel proposed in [14][13]

Only few researchers propose deeper security approaches for MAS development. Mouratidis et al. [85] and Beydoun et al. in [14][13] propose a security-aware general approach for MAS consisting in a framework independent MAS MetaModel (figure II.2.3). The MetaModel regroups/unifies several MAS methodologies and includes security concepts such as *security task* as an inheritance of *agent task* and *to be achieved* by *security goal* which *satisfies* a *security requirement*; *security mechanism* linked to the *agent* concept through the *owned* relationship; and *security constraint* to *restrict* an agent goal. Despite its potential in

representing MAS security requirements and mechanisms, the proposed MetaModel remains at the conceptual level without a MAS platform and tools to support its usage.

Few other approaches [123][124][4] directly extended one of the MAS platforms, the Java Agent DEvelopment platform (JADE), to provide the following four kernel security services to JADE: Security service (authentication); permission service(authorization); signature service(message signature) and encryption service (message encryption). The proposed services are implemented as a plug-in called JADE-S.

In the context of our work, the security concepts proposed in the above mentioned approaches are not enough to express the semantics of SoSSecML security concepts. They lack of appropriate notations to convey the semantics of the *vulnerability*, *pre-condition* and *post-condition* concepts of our MetaModel (cf. I.2.3.2). Therefore, there is a need to enrich MAS MetaModel with some additional security concepts to be able to properly express the semantics of the secure SoS architectures modeled using our DSL and therefore to allow the simulation/execution of these models in MAS.

Executing the described architectures including the security details will allow the unknown emergent behavior, precisely the cascading attack, to manifest as it could happen in real cases. Subsequently, the simulation results will display the details of the discovered cascading attacks. By analyzing those results we will be able to identify the reasons/causes and sequence of actions/operations along with the triggered vulnerabilities (real path) that lead to the cascading attack, as well as its possible effects/results.

II.2.4.2 MAS MetaModel security extension

To express the semantics of the SoSSecML MetaModel security concepts, we extended the MAS MetaModel by the appropriate concepts to convey the semantics of the *vulnerability*, *pre-condition* and *post-condition* as shown in figure II.2.4. We recall below the semantics of these security concepts as previously defined in subsection I.2.3.2:

- 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. The matching mechanism *vul_matching* 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. This mechanism is presented in figure II.2.6 and further detailed later in section II.2.4.4;

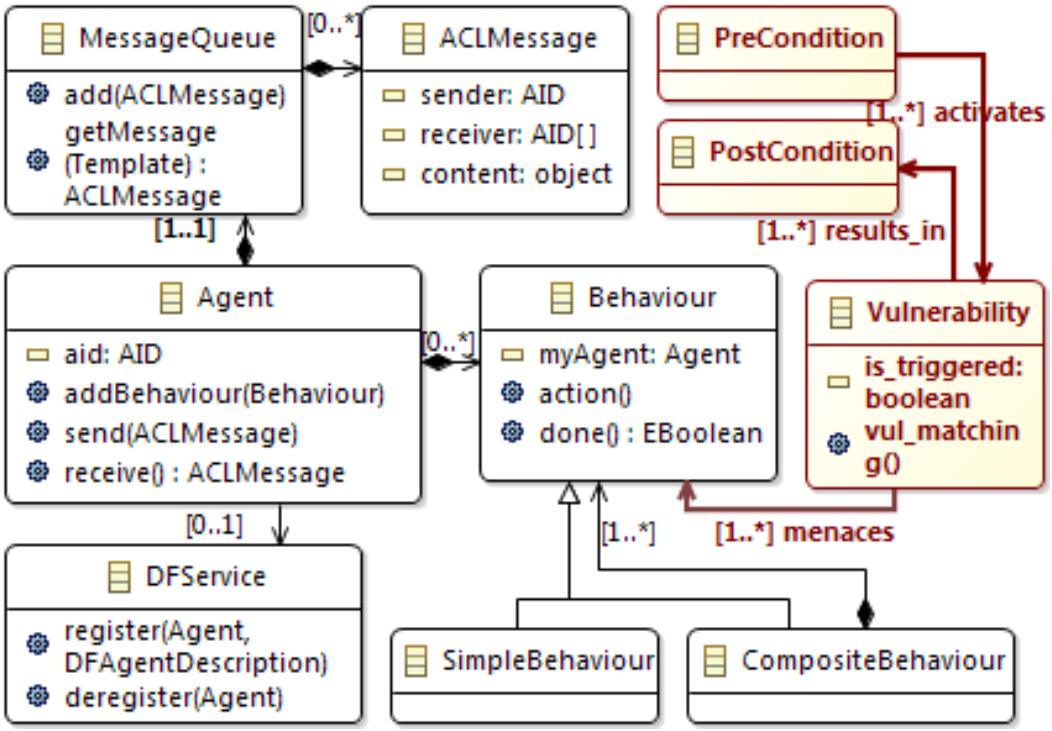


Fig. II.2.4 Part of the MAS MetaModel with the security extension

- *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 et al. in [13] [14] and presented in the previous section II.2.4.1 figure II.2.3. Indeed, the *PreCondition* and the *PostCondition* concepts of our MAS MetaModel extension refines somehow the *Security Constraint* concept of the general MAS MetaModel. However, we adapted the use of the constraints to the context of cascading-attacks, therefore we associate them to the *Vulnerability* concept that we added. In addition we linked these security concepts to the agent behavior/action, not directly to the agent concept, dissimilar to the general MetaModel, in order to align our extended MAS MetaModel to the SoSSecML MetaModel and express the semantics behind these concepts, in particular the fact that we assign the vulnerabilities to the global functionalities/operations defined on the CS interface (cf. subsection I.2.3.2).

Our extension of the MAS MetaModel could be re-used and further extended to model and analyze other security properties and concerns. One extension possibility is the refinement of the *Security Mechanism* of the security-aware general MAS MetaModel (proposed by Beydoun et al. figure II.2.3) to express the SoSSecML security mechanism concept and therefore allow the analysis of the secure SoS architecture after the inclusion of the security mechanisms.

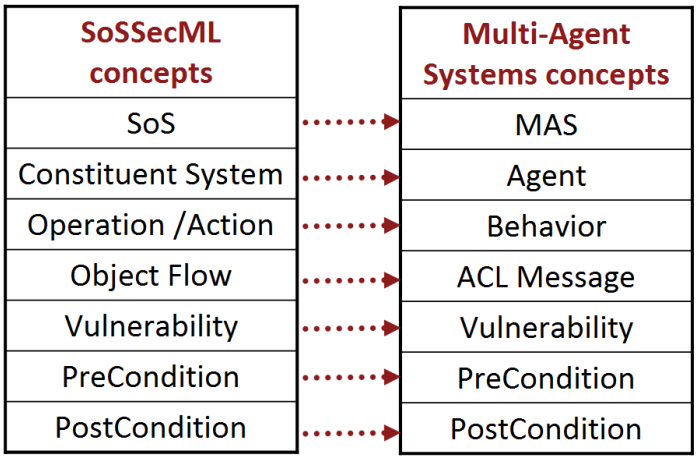


Fig. II.2.5 SoSSecML MetaModel and MAS MetaModel concepts alignment

To conclude this section, our MAS security extension allows the expression of the SoSSecML MetaModel security concept semantics, mainly through the *Vulnerability*, *Pre-Condition* and *PostCondition* extensions. Figure II.2.5 recapitulates the alignments between the SoSSecML MetaModel and the extended MAS MetaModel concepts.

In the following subsections, we detail the implementation of these alignments. We argue the selection and we present the MAS platform to which we implemented the extensions.

II.2.4.3 Java Agent DEvelopment (JADE) MAS platform

A good collection of simulation frameworks/platforms for the execution of MAS can be found in [26][71][18]. Among these, after a comparison study, we adopted the JADE simulation platform to implement our MAS MetaModel security extension, in order to allow the execution the secure SoS architectures modeled using SoSSecML and its graphical editor.

[7] http://sourceforge.net/projects/zeusagent/
[8] http://moise.sourceforge.net/
[9] http://cormas.cirad.fr/indexeng.htm
[10] http://www.madkit.net/madkit/
[11] http://ingenias.sourceforge.net/
[12] http://aosgrp.com/products/jack/
[13] http://jade.tilab.com/

Table II.2.4 MAS platforms

Comparison criteria	**Zeus**[7]	**Moise**[8]	**CORMAS**[9]	**Madkit**[10]	**INGENIAS**[11]	**Jack**[12]	**JADE**[13]
FIPA standard compatibility	✗	✗	✗	✗	✗	✓	✓
Security concerns implementation	✗	✗	✗	✗	✗	✗	✓(JADE-S)
Open source	✓	✓	✓	✓	✓	✗	✓
Last update	2013	2014	2012	2014	2013	2009	Dec. 2015
Popularity	Medium	Medium	Medium	Medium	Medium	High	Very high
Documentation and ease of use	Medium	Medium	Medium	Medium	Medium	High	High

We compared the most popular MAS platforms from [71][18][26], based on several criteria that we consider important in the context of our work, mainly to implement and execute the extended MAS MetaModel concepts that will express the semantics of our DSL. Following are the platform selection criteria:

- FIPA standard compatibility: This is the essential criteria for the platform selection. Indeed, the platform should comply to the most widely adopted MAS standard, FIPA, on which we based our similarity analysis and we founded our semantic correlation/mappings (section II.2.2);
- Security concerns implementation: Even if our study (subsection II.2.4.1) shows that the security in MAS platforms is relatively concentrated on the agent communications and they lack security concepts to express the semantics of the SoSSecML security, however we included the security concern in our comparison so that may enable us in future work to consider the secure communications and trust between CSs/agents;
- Open source: For all the benefits that open source tools support, such as collaborative development from multiple independent sources and long term sustainability[14] and cost savings, as well as for our need of extending the platform to implement our MAS security extensions;
- Last update: To make sure that the platform is recently maintained, updated and improved;

[14]Open source software: https://en.wikipedia.org/wiki/Open-source_software

- Popularity: This criteria describes the interest that users have in the platform. As reported in [12][71][18][26], JADE has high popularity in terms of the high number of users compared to the other platforms;

- Documentation and ease of use: We consider these two criteria essential for the time development concern, which is a crucial consideration in a three year Ph.D. Based on our evaluation, JADE seemed well documented compared to the other platforms. In addition, being implemented using one known language (Java) makes it easier to manipulate and extend JADE, on the contrary to the other platforms implemented in several languages (such as Java and JAL for the Jack platform; Java, C++ and Python for Madkit; Java and XML for INGENIAS); which is an additional reason for the popularity of JADE.

Our comparison presented in table II.2.4 shows that the Java Agent DEvelopment (JADE) platform is the only open source framework which is FIPA-compliant and where security concerns are considered. It has been classified, in recent surveys, as the most popular and the leading open source agent-based framework, widely used and actively maintained [71][12]. JADE is an open source framework implemented in Java for the development of distributed MAS, based on peer-to-peer communication architecture. It implements the FIPA specifications described in section II.2.2, beyond that "JADE is generally considered as the leading FIPA-compliant open source technology available today" [12]. Therefore we adopt the JADE MAS platform and we extend it, as described in the next subsection, to execute the secure SoS architectures designed using SoSSecML.

II.2.4.4 Implementing the MAS MetaModel security extension in JADE

We used the latter JADE platform existing concepts to express the semantics of SoSSecML structural and behavioral concepts (as detailed in the following section). Moreover, knowing that in JADE there are no security concepts pertinent to our needs, we extended JADE to implement the extended MAS MetaModel and therefore cover SoSSecML security concepts: *Vulnerability, PreCondition* and *PostCondition*. In addition we implemented the vulnerability matching mechanism *vul_matching* that 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 CS operations. To implement these security extensions, we extended the JADE simulator by defining the following 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

Algorithm *vul_matching* **- Vulnerabilities pre and post conditions matching**

Input: vul // Current vulnerability
L1[] // List of pre-conditions of the current vulnerability *vul*
L2[] // List of post-conditions of the vulnerabilities triggered on previously executed operations/interactions and received within the ACL message
Output: is_triggered // Boolean variable
log // log
Begin
For each (pre_condition ϵ L1[]) {
For each (post_condition ϵ L2[]) {
if (pre_condition.evaluate(post_condition) = = true) {
vul.is_triggered = true;
log ([CS name] INFO sec.Vulnerability-behaviour: Security Breached on '*vul*' --'*the matched pre-post condition*' *is_triggered*) }}}
End

Fig. II.2.6 Vulnerability matching algorithm

value (false by default) and a *vul_matching* mechanism. The result of this matching mechanism is reported in the JADE *log_file*;

- *public abstract class PreCondition<L>*: To specify the list of pre-conditions that activate 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* (figure II.2.6): 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 match 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 is to discover the

cascading attacks resulting only from the execution of CSs operations that interact one with another, in contrast to verifying all possible combinations, avoiding in this way the combinatorial explosion of existing attacks analysis approaches.

After describing the target JADE platform and the extensions we needed to simulate the SoS architectures and their security issues, we present in the next section the implementation of the semantic mapping (the Model-To-Text mapping rules) that we defined to implement the alignments between SoSSecML and MAS concepts, and therefore to permit the automatic mapping between the modeling and simulation phases.

II.2.5 SoSSec generator tool: Model transformations from SoSSecML to the extended MAS platform (JADE)

In order to gain the maximum engineering value from MDE methods, it is required to have well-defined mappings between the models and their realizations [92]. A well defined and relatively easy mapping between the modeling and execution concepts increases the likelihood that the domain concepts (in our case secure SoS) will be captured accurately [115].

In line with the MDE approach and as described in section II.2.3, SoSSecML semantics are expressed by assigning the semantics of the semantic domain elements (extended MAS MetaModel) to those of the syntactic elements (SoSSecML MetaModel). In this section, we introduce the semantic mapping and we present the SoSSec MetaTools that we defined to perform the mapping in an automatic way. Afterwards, we detail the implementation of these mappings.

II.2.5.1 MDE Meta-Modeling approach for DSL definition: The semantic mapping

The semantic domain of a DSL defines the meaning that the concepts of the DSL denote[115]. One of the main advantages of following MDE to define a DSL is the fact of defining platform-independent languages where the syntactical elements are defined independently from the semantic domain.

Following MDE, one effective way to define the semantic domain of a DSL is through the semantic mapping towards the precise semantics of an existing programming language[72]. The semantic transformation/mapping links the abstract syntax to the semantic domain. It is a technique for the interpretation of an abstraction of a system (such as the secure SoS

architecture) into a semantically richer model (such as MAS) by using a specified set of model transformations [72]. *Model transformations* are one of the key mechanisms within MDE, they define a set of rules that allow the mapping from a source model to a target model. Each transformation rule describes how each concept in the source language will be mapped to a concept in the target language. Among the model transformation types, the one on which we are focusing here is the Model-To-Text transformations (code generation) [27]. We chose using MTT (instead of Model-to-Model transformations for example) because this is a "vertical" transformation, from the MetaModel of SoSSecML, at a higher level of abstraction, towards the extended JADE Java code, which is at an implementation/simulation level of abstraction.

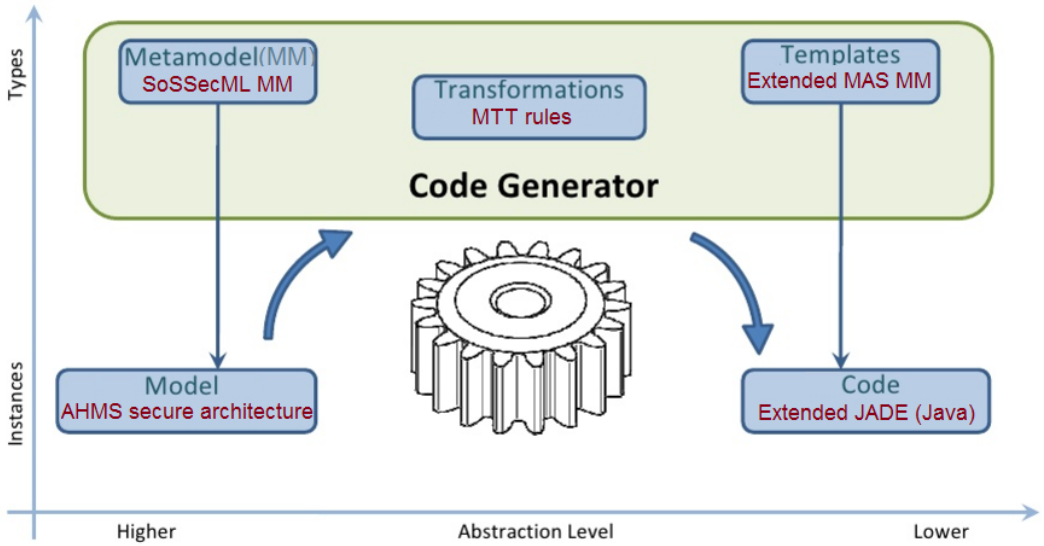


Fig. II.2.7 Conceptual map for code generation from [82]

As described in figure II.2.7, a *code generator* defined through *transformations* (such as MTT transformation rules) translates the modeling elements (a *model* conform to its *MetaModel*) into an executable representation (an executable *code* conform to its *template*). In our work, we defined a set of MTT transformation rules to generate the code from the secure SoS architectures conform to the SoSSecML MetaModel to the Java executable classes of the extended JADE platform, which is conform to the FIPA specifications for MAS and our extension of the MAS MetaModel.

II.2.5.2 SoSSec generator tool construction process

Figure II.2.8 exhibits the SoSSec generator tool construction process defined following the MDE approach and using modifiable, extensible and reusable MetaTools. The process

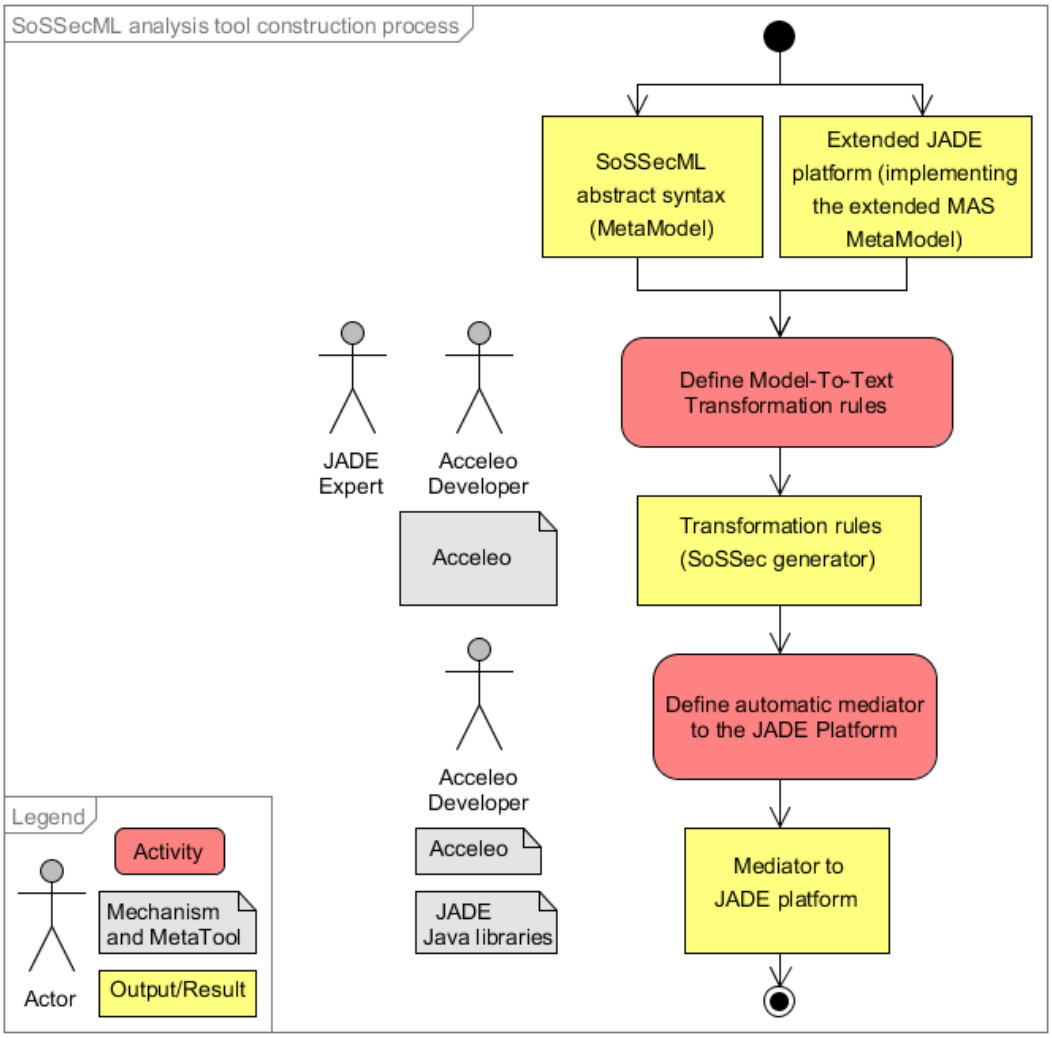


Fig. II.2.8 SoSSec analysis tool construction process

describes the steps, actors and meta-tools used to define the generator and the link to the extended JADE simulation platform. Having the SoSSecML MetaModel and extended JADE platform that implemented the extended MAS MetaModel, our Acceleo developer and JADE expert - M. Tarek AL KHALIL[15] - implements a set of Model-To-Text (MTT) transformation rules to map the concepts (generate code) from the secure SoS modeling concepts to the Java classes of the extended JADE platform. Afterwards, the developer uses the previously defined MTT rules in addition to some minor Java development (in particular adding the JADE libraries into the Acceleo project) to make the link with the JADE platform where the models will be executed.

[15]M. AL KHALIL was a master student at the Antonine University, Lebanon. He implemented the SoSSec MTT generator in the context of his six month master internship realized under the supervision of Mme. EL HACHEM

In the following subsections we introduce the basics of the semantic mapping paradigm and the Acceleo language, then we present the implemented set of MTT rules and the logging mechanism that we defined to capture the simulation results.

II.2.5.3 Acceleo model transformation language

To define our code generator, we choose to use template-based approaches for the automatic code generation. Our choice is due to the maturity and well-establishment of the template-based approaches in software engineering [19], as well as the availability of corresponding powerful, mature and free tools (template engines). As shown in figure II.2.9, in template-based approaches, 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.

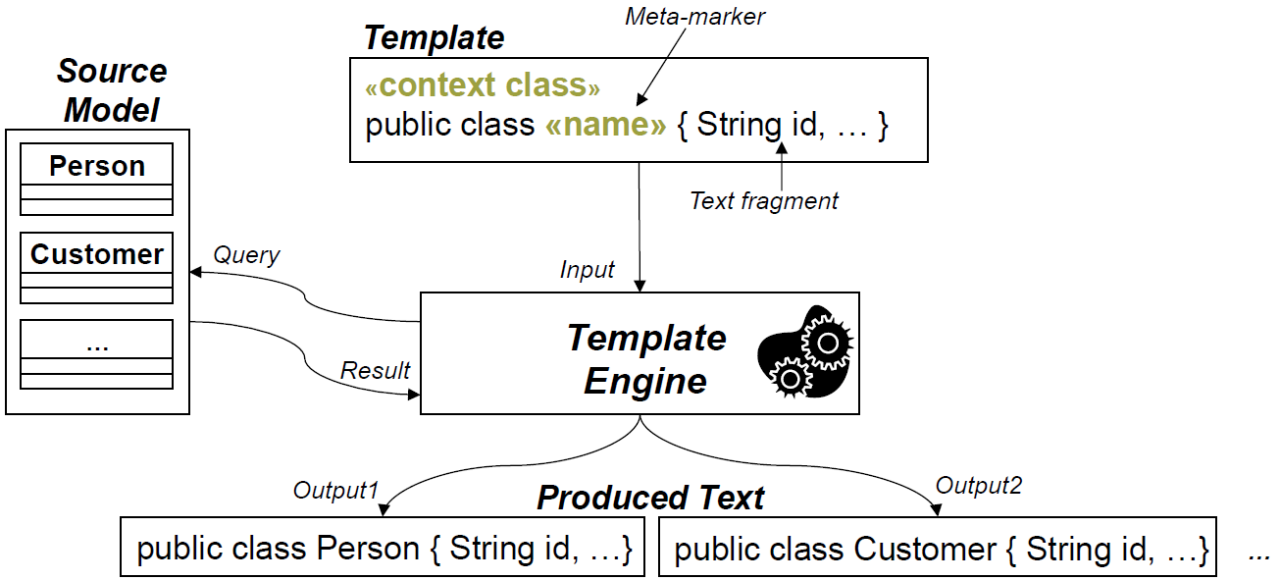


Fig. II.2.9 Template-based approaches from [19]

Numerous Eclipse-based template languages and their corresponding tools (template engines) are available, such as the Java Emitter Templates (Jet[16]), (Xpand[17]) and Acceleo[18]. For our work, we used Acceleo, an Eclipse-based tool developed following MDE. We selected Acceleo among the others essentially for the following features: Acceleo is a mature implementation of the Object Management Group MTT transformation standard[19], it is

[16]https://eclipse.org/articles/Article-JET/jet_tutorial1.html
[17]http://wiki.eclipse.org/Xpand
[18]http://www.eclipse.org/acceleo/
[19]http://www.omg.org/spec/MOFM2T/1.0/

an open source tool, recently used by the community, maintained by the developers and relatively well documented and easy to use. It offers powerful support by offering an editor, debugger, profiler and traceability between model and code [19].

In the following subsection, we present the MTT rules (templates) that we define using Acceleo, to map the secure SoS source model elements to the JADE Java classes.

II.2.5.4 SoSSec generator tool: Model-To-Text transformation rules

As previously described in subsection II.2.5.1, with respect to MDE, each MTT transformation rule defines the mapping from a concept/model element in the source model (in our case SoSSecML concepts) to the corresponding concept in the target platform (in our case the JADE platform).

Based on the set of similarities between the SoSSecML concepts and those of the extended MAS concepts (cf. section II.2.3), we present in what follows our MTT rules implemented using Acceleo to allow the mapping from the secure SoS models designed using SoSSecML to the extended JADE platform classes (figure II.2.5):

- **CS to Agent**: The *Constituent System* (CS) concept of SoSSecML is mapped to the *agent* class of the JADE platform (*jade.core.Agent class*);

  As we can see in figure II.2.10, this first MTT generates 1) the name of the class and 2) the java package of the class, 3) it sets the name and path to the class file and 4) the import statements for the packages that are used in the class, 5) it also generates the class body and 6) 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:

- **Operation to Behavior**: The CS *operation* concept of SoSSecML is mapped to an instance of the agents *behavior* class (*jade.core.behaviour.Behaviour*) taking into account the following types of behaviors (TickerBehavior, OnShotBehavior, CyclicBehavior);

- **Flows to Messages**: The interactions between different operations defined on different CS interfaces and modeled using the *ObjectFlow* concept of SoSSecML are mapped to agents interactions represented by message exchange services, specifically a *service helper* (jade.core.ServiceHelper) called *TopicManagementHelper* (*TopicManagementHelper extends ServiceHelper*), which handles the operations related to ACLMessage;

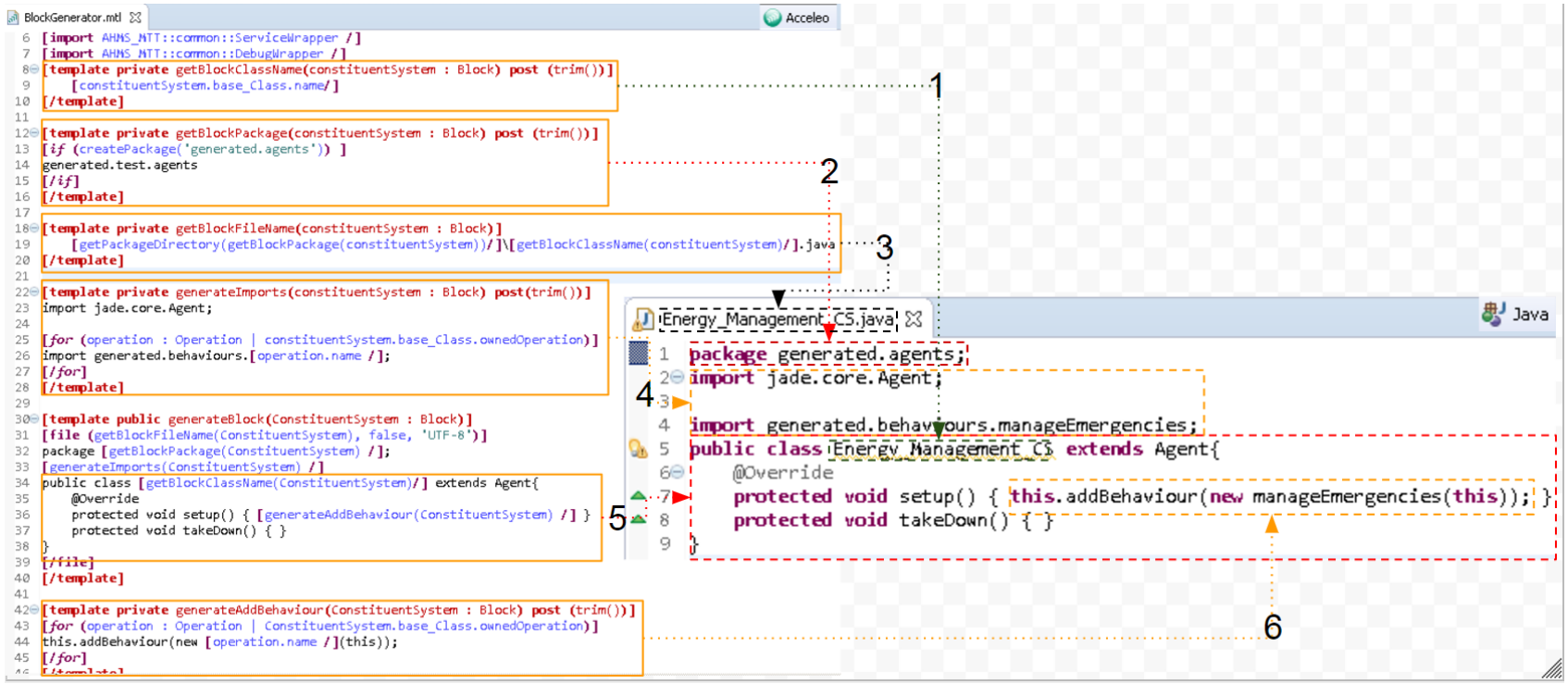


Fig. II.2.10 Acceleo MTT rule to map the CS concept to the Agent class

- **Vulnerability to Vulnerability**: The *vulnerabilities* described using SoSSecML on the CS operations are mapped towards the JADE *Vulnerability* class we introduced and assigned to the corresponding agent behaviors;
- **Pre-condition to pre-condition**: This rule is to settle the list of pre-conditions for each vulnerability;
- **Post-condition to post-condition**: Since we are interested only in discovering the sequence of vulnerabilities that are triggered because of the CS interactions, this rule is defined with respect to the *vul_matching* mechanism, in a way to map only the post-conditions of the *triggered* vulnerabilities linked to the *executed* operations. Thus, these post conditions are sent within the messages described in the third rule Flows to Message.

These transformation rules constitute the automatic generator that takes as input a secure SoS architecture and produces the corresponding Java code. Afterwards, the generated classes are executed in the JADE platform, and the results are logged, as described in the following section.

II.2.6 JADE simulation and log file creation

After defining the MTT rules, the generated code (JADE Java classes) are directly executed in the JADE platform through an automatic link that we created between the Acceleo tool and the extended JADE platform, by specifying the JADE project as *target* when defining the Acceleo running Eclipse *project*.

Therefore, by giving these rules, together with the secure SoS architecture models (defined using SoSSecML) as input to the Acceleo generator, the produced Java classes are 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 our analysis, we seeked 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*[20]. Log4J is a Java-based logging utility allowing customized log levels and offering Asynchronous Loggers.

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

```
283 [template private  generateActionMethod(behaviourType : String, behaviourName : String)]
284 @Override
285 public void action() {
286 [if (behaviourType.equalsIgnoreCase('CyclicBehaviour'))]
287     ACLMessage msg = myAgent.receive(_mt);
288     if(msg != null) {
289         LOG.info(new LogMessage([behaviourName /].class, "Receive Message from: " + msg.getSender()));
290
291         //check for security issues
292         Message contentObjectMessage;
293         try{
294             contentObjectMessage = (Message) msg.getContentObject();
295             LOG.info(new LogMessage([behaviourName /].class, "ContentMessage: " + contentObjectMessage.toString()));
296
297             Vulnerability['['.toString() /][']'.toString() /] receivedVulnerabilities = contentObjectMessage.getVulnerabilities();
298             for(Vulnerability vul : this._vulnerabilities){
299                 for(Vulnerability receivedVul : receivedVulnerabilities){
300                     if(!receivedVul.isTriggerd()) continue;
301                     if(vul.evaluate(receivedVul.getPostConditions())){
302                         vul.setTriggerd(true);
303                         LOG.info("Security breached due to the '" + receivedVul.getName() + "' postConditions.");
304                         LOG.info(new LogMessage(this.getClass(), "\n^^^^^^^^^^^^^\nSECURITY ISSUE\n^^^^^^^^^^^^^"));
305                     }
306                 }
307             }
```

Fig. II.2.11 Personalized log file

As shown in figure II.2.11, the custom log file (*LOG.info()*) issued from the simulation of the secure SoS architecture will reveal the triggered vulnerabilities, the pre-post conditions matching that triggered the vulnerabilities (*receivedVul.getname()* + *postConditions*), as well as the interactions (CSs and operations - *behaviourName, ContentMessage* and *Sender*) to which these vulnerabilities belong. Having these information captured in the log files, the architects and security experts will be able to analyze the real structure and 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.

II.2.7 SoSSec utilization process analysis phase

To complete our SoSSec utilization process that we start describing in section I.2.6, we present in figure I.2.14 the full utilization process (published in [53]) including the in addition to previously described modeling phase, the analysis phase to guide the use of our SoSSecML DSL and tools. This process describes the activities that should be followed when using SoSSecML, along with the corresponding actors and required tools, to model and analyze secure SoS architecture.

Having the SoS secure architecture represented in one model with different views/diagrams (output from the modeling phase), the afterward step is to execute these architectures and analyze the results. The execution/simulation of these models allows the analysis of the SoS interactions and the discovery of the emergent cascading attacks resulting from these

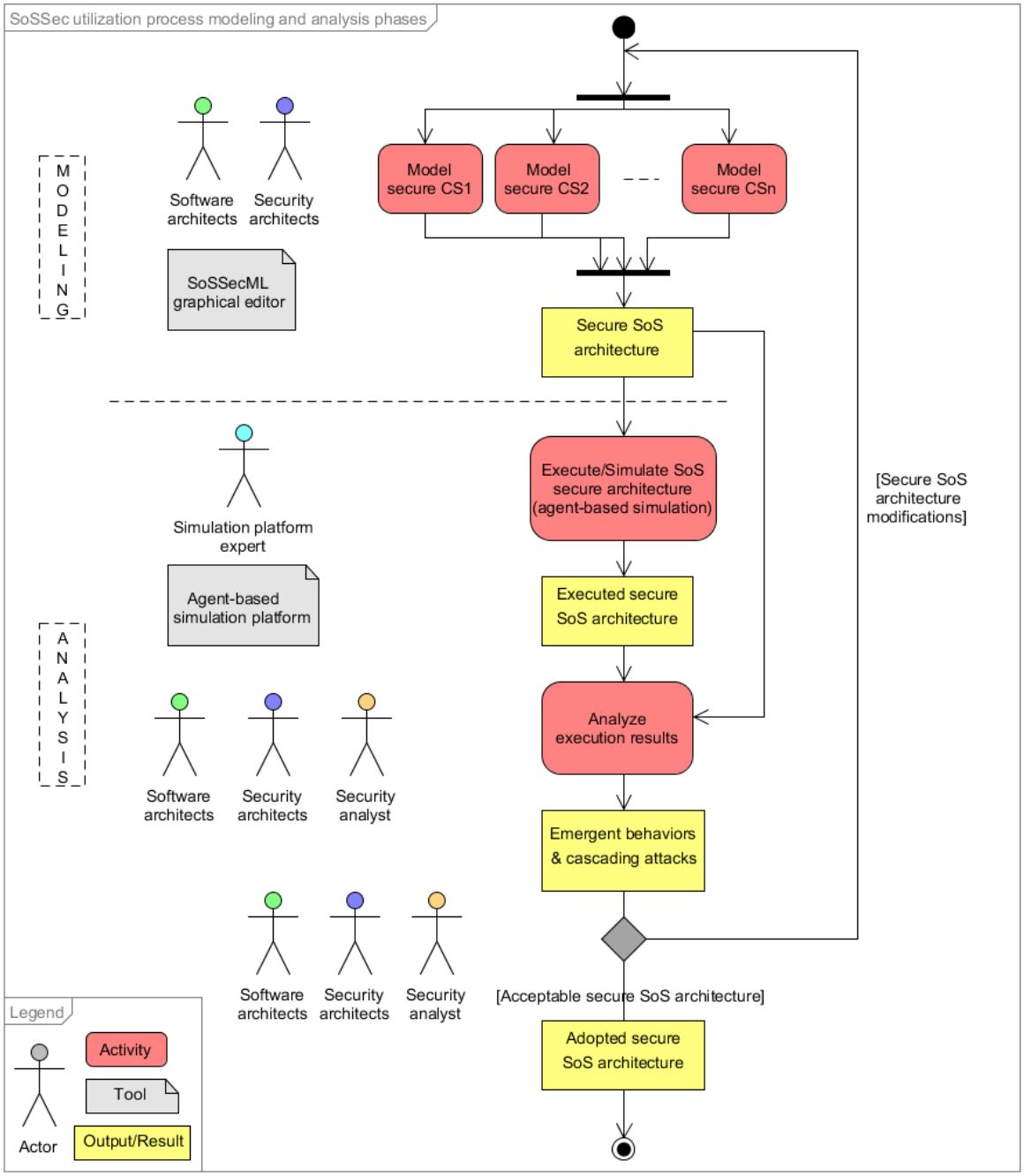


Fig. II.2.12 SoSSec utilization process - Modeling and Analysis phases

interactions. For this analysis phase, the simulation platform expert applies a set of **M**odel-**T**o-**T**ext (MTT) transformation rules that we defined, on the previously designed secure SoS architecture to generate the corresponding code. The resulting code is then executed in the agent based simulation platform that we extended to handle the security concepts. Afterwards, the execution results are analyzed to discover the emergent cascading attacks.

Consequently, software and security architects and security analysts decide whether the secure SoS architecture is acceptable in terms of compliance to the needed security level, or there is a need to modify it and/or define some security mechanisms and go through the whole process again.

Figure II.2.13 details the analysis phase of the process. As shown in the figure, once the SoS secure architecture is described, the Java Agent DEvelopment framework (JADE) expert applies the MTT rules defined using Acceleo to generate the corresponding code (Java classes) to the existing multi-agent simulation platform, in particular JADE. Then, the resulting Java classes are executed/simulated to observe CS/agent interactions and discover the vulnerabilities that are triggered and connected through these interactions to form the unknown cascading attacks. The log file describing the results is then sent to the software and security architects as well as to the security analyst to be further analyzed.

At this stage, having the execution results (log file), the analysis could be realized qualitatively and/or quantitatively. Indeed, for a *qualitative analysis*, the software and security architect and the security analyst analyze the log files, and manually depict the triggered vulnerabilities on the CS operations, as well as the unknown sequence of successive security breaches leading to each possible cascading attack (cascading attack path). On the other hand, a *quantitative analysis* could include computational metrics, score calculation and visualization tools to help the architects and analysts interpret the log file results.

This analysis process leads to the prediction/discovery of the possible unknown cascading attacks (emergent behavior) related to the analyzed secure SoS architecture. Consequently, based on these results, the software and security architect and the security analyst will decide if the current secure SoS architecture is acceptable and meet their expectations in term of security or it should be modified to avoid the discovered cascading attacks. For the latter choice, corrective actions (such as security mechanisms, patches for the triggered vulnerabilities) should be introduced to the initial secure SoS architecture, and the whole process should be restarted again: The new architecture alternative should be modeled, executed and analyzed to discover new possible emergent behaviors and security cascading attacks.

After following the modeling phase of this utilization process to model the AHMS smart building case study secure architecture in subsection I.3.4.2, we followed the previously described analysis phase of this process to **analyze** the AHMS secure architecture as detailed in the next chapter.

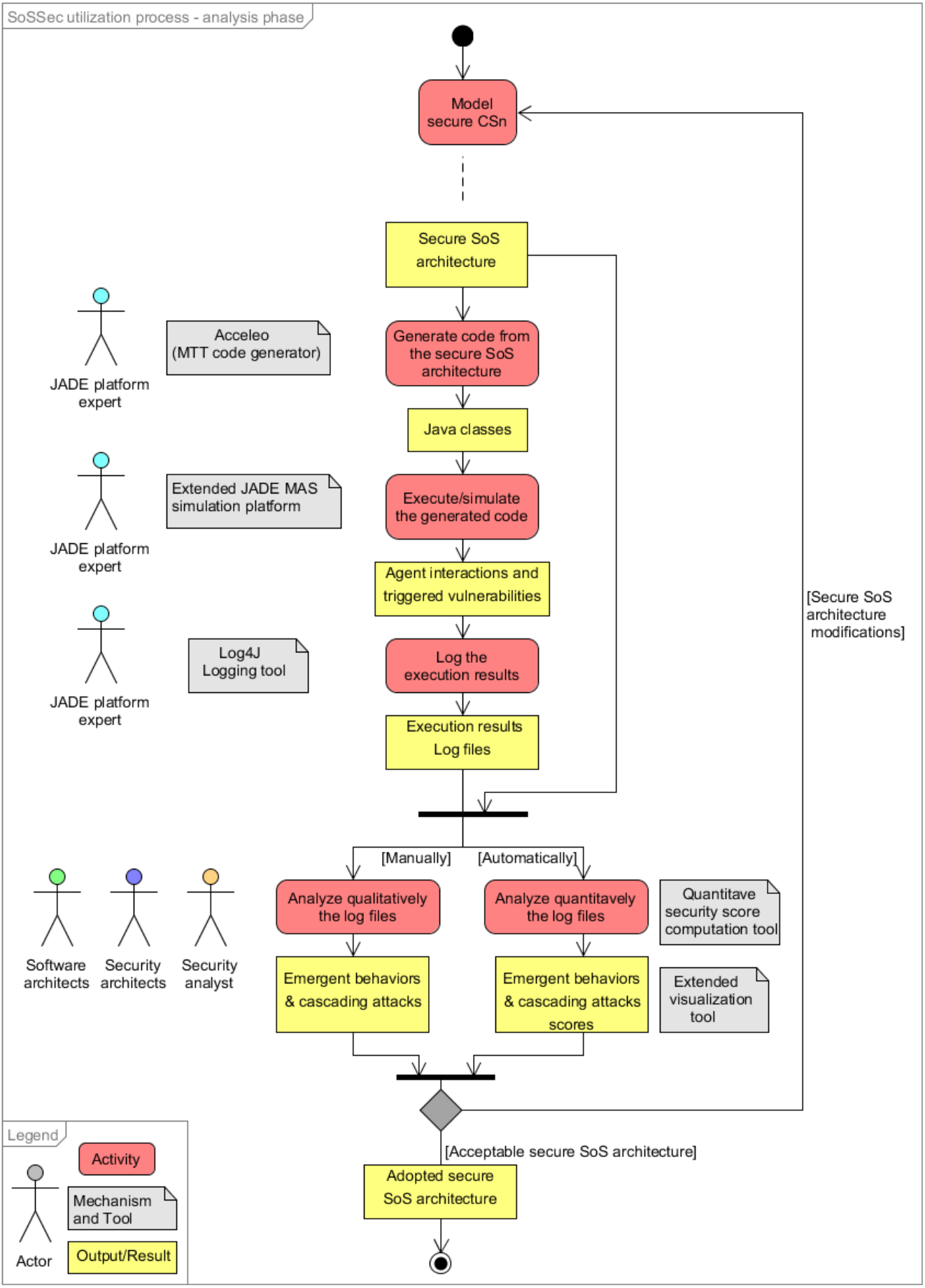


Fig. II.2.13 SoSSec utilization process - Analysis phase

II.2.8 Conclusion

To conclude, we presented in this chapter the semantics of our SoSSecML DSL, inherited from MAS semantics. To define these semantics we defined a security extension to a MAS MetaModel and we aligned the concepts of this extended MetaModel to those of the SoSSecML MetaModel. Then, we selected a MAS platform: the Java Agent Development Platform (JADE) and we implemented the MAS MetaModel extensions into JADE. Afterwards we used the semantic mapping mechanism of MDE (MTT rules for code generation) to implement the mappings between SoSSecML and extended MAS MetaModel concepts. By implementing these mappings, based on the semantic similarities between both SoS and MAS domains, we offered a powerful simulation and analysis paradigm for the secure SoS architectures, addressing by that the simulation challenge stated in the general introduction (problem 3). Indeed, executing the described architectures including the structural, behavioral and security details, allows the unknown emergent behavior, precisely the cascading attack, to manifest as it could happen in real cases. Therefore, the simulation results display the details of the discovered cascading attacks. By capturing and analyzing those results, we are able to identify the reasons/causes and sequence of actions/operations along with the triggered vulnerabilities (real path) that lead to the cascading attack, as well as its possible effects/results, answering by that our second research question RQ2. One additional advantage of this analysis is the vulnerability matching mechanism that we defined to discover the cascading attacks resulting only from the execution of CSs operations, avoiding in this way the combinatorial explosion of existing attacks analysis approaches. Indeed, given a SoS global goal(s), the CSs/agents of this SoS/MAS are supposed to interact, to use their global functionalities/operations/behaviors defined on their interfaces, to accomplish this/these global goal(s). In this context, the SoSSec matching mechanism tests if a vulnerability should be triggered or not, only for the CS/agent interactions needed for the realization of the SoS global goal(s). In consequence, our SoSSec mechanism allows the discovery of the possible sequence of triggered vulnerabilities resulting only from the execution of CSs operations/agent behaviors. These contributions were published in our paper [55].

In the next chapter, we apply the proposed contributions to analyze the secure architectures of the AHMS smart building.

Chapter II.3

Case Study: Adelaide Health and Medical School (AHMS) smart building secure architecture analysis using the SoSSec method

II.3.1 Introduction

As previously declared in the first part of this thesis (cf. chapter I.3) we applied it on a real-life case study, the Adelaide Health and Medical School (AHMS) smart building SoS, to show how our SoSSec method can be used. This work was accomplished during a research visit to the university of Adelaide, under the supervision of Prof. Ali BABAR and in collaboration with the AHMS team (mainly the energy manager from the infrastructure branch: Mr. KENJLE) as detailed earlier in section I.3.3. In this chapter, we present the application of the SoSSec method *analysis* phase to execute and analyze the previously modeled secure architecture of the AHMS case study.

After modeling the AHMS smart building secure architecture, the next phase of the SoSSec method consists in executing this architecture in our extended MAS platform using the SoSSec generator defined in the previous chapter II.2. Therefore, we apply in this chapter our SoSSec method (SoSSecML language, tools and processes) to execute and analyze a secure SoS architecture, the AHMS secure architecture in our case study (cf. figures I.3.7, I.3.10, I.3.11, I.3.12), in order to discover the possible cascading attacks, starting from the elementary minor vulnerabilities we previously modeled, and building upon the

interactions that may trigger the vulnerabilities once their pre-conditions are met as detailed in the previous chapter.

In section II.3.2, we followed the analysis phase of the SoSSec utilization process presented in the previous chapter to analyze the AHMS smart building secure architecture and discover the possible emergent cascading attacks. In section II.3.3 we discuss limitations of our case study, and we conclude the chapter in section II.3.4.

II.3.2 AHMS secure architecture analysis using SoSSec

We followed the detailed SoSSec utilization process presented in the previous chapter (cf. section II.2.7, figure II.2.13) to execute and analyze our AHMS secure architecture (output from the modeling phase).

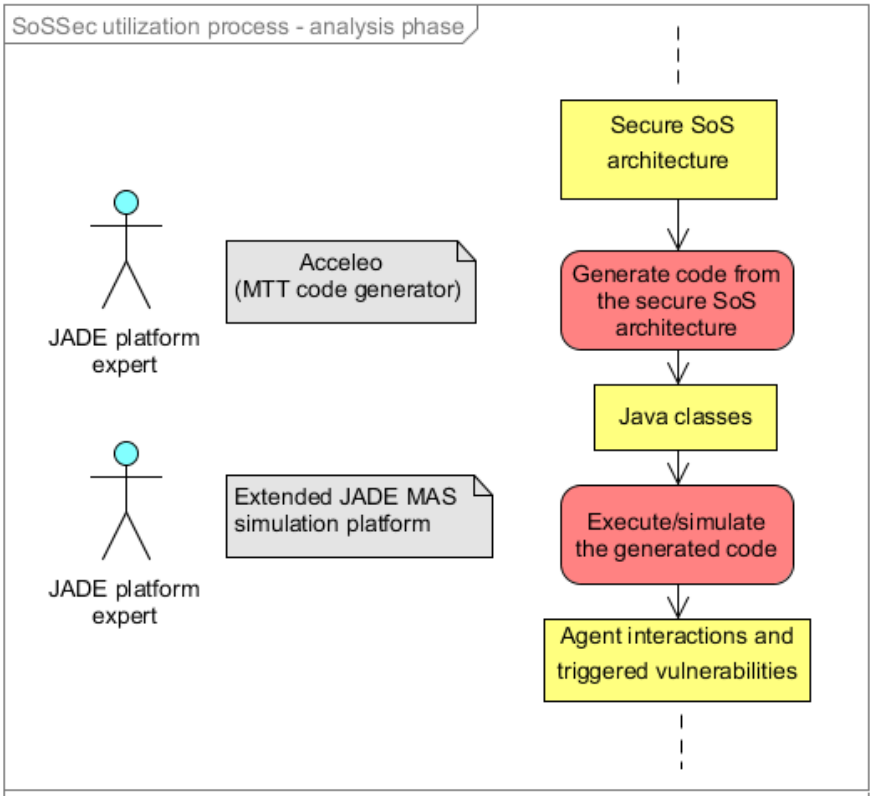


Fig. II.3.1 SoSSec Utilization Process - Analysis phase - Code Generation

II.3.2.1 Code generation and AHMS secure architecture execution

As shown in the snippet II.3.1 of the code generation and architecture execution activities described in our detailed utilization process, having the *secure architecture of the AHMS*,

the *JADE expert* of our SoSSec research team (M. AL KHALIL, cf. section I.3.3) took the AHMS secure architecture model as input in the *Acceleo tool*. He applied on it the set of **M**odel-**T**o-**T**ext (MTT) transformation rules (defined in the previous chapter II.2.5.2) that he implemented in Acceleo. This semantic mapping (application of the MTT rules) results in Java classes (code) corresponding to the AHMS model and executable in the extended JADE MAS platform. Therefore, the generated Java code corresponding to the AHMS models is then executed in the extended JADE simulation platform. This simulations enables the observation of the AHMS CS/agent interactions and the discovery of the vulnerabilities that are triggered and connected through these interactions to form the unknown cascading attacks.

By applying this phase of the utilization process on the AHMS case study, we confirmed the smooth applicability of our SoSSec method for mapping the concepts from a secure SoS architecture modeled using our SoSSecML language to Java code executable in the MAS platform that implements our MAS security extension. Indeed our JADE expert, testified that the mappings defined in our SoSSec method allowed the (semi)automatic mapping and the execution of the secure AHMS models (secure architecture of the SoS) in the extended JADE platform, indeed all what he (the user) did was to import the models to Acceleo and to execute the pre-defined SoSSec MTT rules. The generated code was automatically executed in the extended JADE as result of the link that we made between Acceleo and the platform. Another user[1] having only basic knowledge of Acceleo, employed our SoSSec method (language and tools) in the context of her research. She also affirmed that, following this part of the utilization process, she was able to generate and execute the corresponding code for her SoS models.

II.3.2.2 Logging the AHMS secure architecture execution results

As shown in the snippet II.3.2 of the logging activity described in our detailed utilization process, the resulting agent interactions and triggered vulnerabilities were automatically logged in a customized log file (cf. subsection II.2.6).

[1]Mme. Ouijdane GHIZA was a master student at the University of Manouba, Tunisia. She used the SoSSec method in the context of her six month master internship realized under the supervision of M. CHIPRIANOV and Mme. EL HACHEM

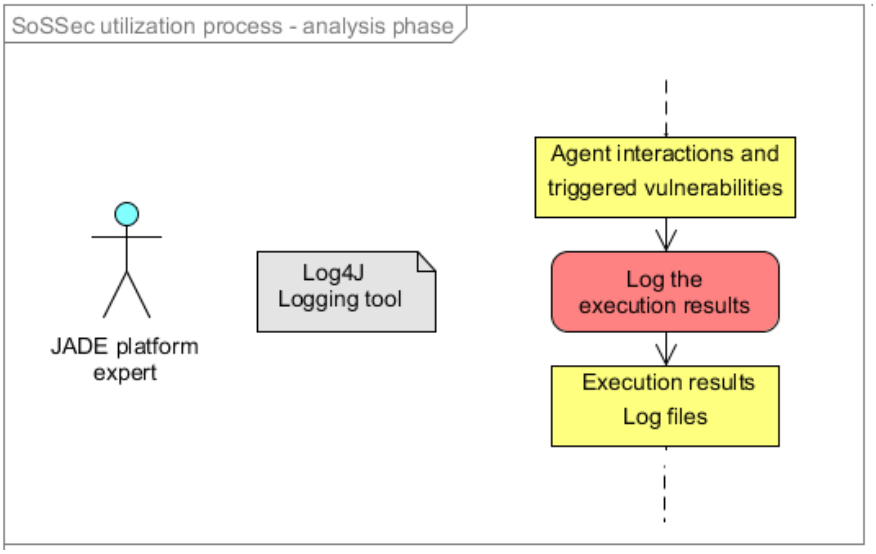


Fig. II.3.2 SoSSec Utilization Process - Analysis phase - Logging

AHMSLogFile.txt

```
00:55:47.793 [BMS] INFO  src.generated.behaviours.automaticControlHVAC
              - Extends: [class jade.core.behaviours...]-- Initialising Behaviour
00:55:55.133 [Lighting] INFO  src.generated.behaviours.automaticControlLights
              - Extends: [class jade.core.behaviours...]-- Initialising Behaviour
00:56:02.715 [SmartGrid] INFO  src.generated.behaviours.manageEnergy
              - Extends: [class jade.core.behaviours...]-- Initialising Behaviour
...
-- Vulnerabilities - Vulnerability manageEnergy-Vul-WeakAuthentication (isTrigged: true)

00:56:02.825 [BMS] INFO  src.sec.Vulnerability  ------- automaticControlHVAC :
              Security Breached on automaticControlHVAC-Vul-DirectoryTraversal
              -- the pre-condition 'automaticControlHVAC-Post-AccessPasswordInformation' has been triggered.
             The post conditions which resulted in this breach are: manageEnergy-Post-AccessPasswordInformation
...
00:56:02.827 [BMS] INFO  src.sec.Vulnerability ------- automaticControlHVAC :
              Security Breached on automaticControlHVAC-Vul-PrivilageEscalation
              -- the pre-condition 'automaticControlHVAC-Post-RemoteAccess' has been triggered.
             The post conditions which resulted in this breach are: manageEnergy-Post-RemoteAccess
...
-- Vulnerabilities --
   Vulnerability automaticControlHVAC-Vul-DirectoryTraversal (isTrigged: true)
   Vulnerability automaticControlHVAC-Vul-PrivilageEscalation (isTrigged: true)
...
    00:56:02.875 [Lighting] INFO  src.sec.Vulnerability  ------- automaticControlLights :
              Security Breached on automaticControlLights-Vul-CrossSiteScripting
              -- the pre-condition 'automaticControlLights-Post-ByPassAuthentication' has been triggered.
             The post conditions which resulted in this breach are: automaticControlHVAC-Post-ByPassAuthentication
...
-- Vulnerabilities - Vulnerability automaticControlLights-Vul-CrossSiteScripting (isTrigged: true)
...
```

Fig. II.3.3 AHMS smart building simulation results - Log file snippet

Figure II.3.3 exhibits a snippet of the log files containing the AHMS secure architecture execution results. We can see:

- The agents, their behaviors and interactions, in particular the *BMS*, *Lighting* and *SmartGrid* agents ;
- The triggered vulnerabilities, such as *WeakAuthentication*, *DirectoryTraversal*, *PrivilageEscalation* and *CrossSiteScripting*;
- The pre-post conditions matching that triggered the vulnerabilities (security breaches) such as *AccessPasswordInformation*, *RemoteAccess* and *BypassAuthentication*;
- The agent behaviors/CSs operations to which these vulnerabilities are related, in particular *manageEnergy* and *automaticControlHVAC*.

The application of this phase of the SoSSec method on the AHMS case study, following the utilization process, shows that the automatic logging activity helps identifying the main elements of the simulation results such as the agent/CS interactions, the triggered vulnerabilities, why and how they have been triggered as well as the operations and CSs to which these vulnerabilities are linked.

In addition, the log file proves the operability of the SoSSec vulnerability matching mechanism (vul_matching) in triggering the vulnerabilities when there is a match between the pre-conditions of the vulnerabilities "i" (linked to the operation currently in execution) with the post-conditions of the previous vulnerabilities (linked to an executed operation). The analysis of these logged information will allow the discovery of the possible sequence of vulnerabilities that might be triggered due to the execution of the modeled secure SoS architecture.

The log file describing the execution/simulation results was then sent by the JADE expert of our team to the software and security architects as well as to the security analyst of our team (Mme. EL HACHEM) to be further analyzed.

II.3.2.3 Cascading attacks prediction: Qualitative analysis of the AHMS execution result log files

Even if our SoSSec method envisage both quantitative and qualitative analysis of the log files, however in this case study we limited our work to the qualitative analysis.

As shown in the snippet II.3.4, having the log file, the *software and security architects* of the SoSSec research team (Ms. EL HACHEM) qualitatively analyzed and interpreted the results to extract the emergent cascading attacks (sequence of vulnerabilities

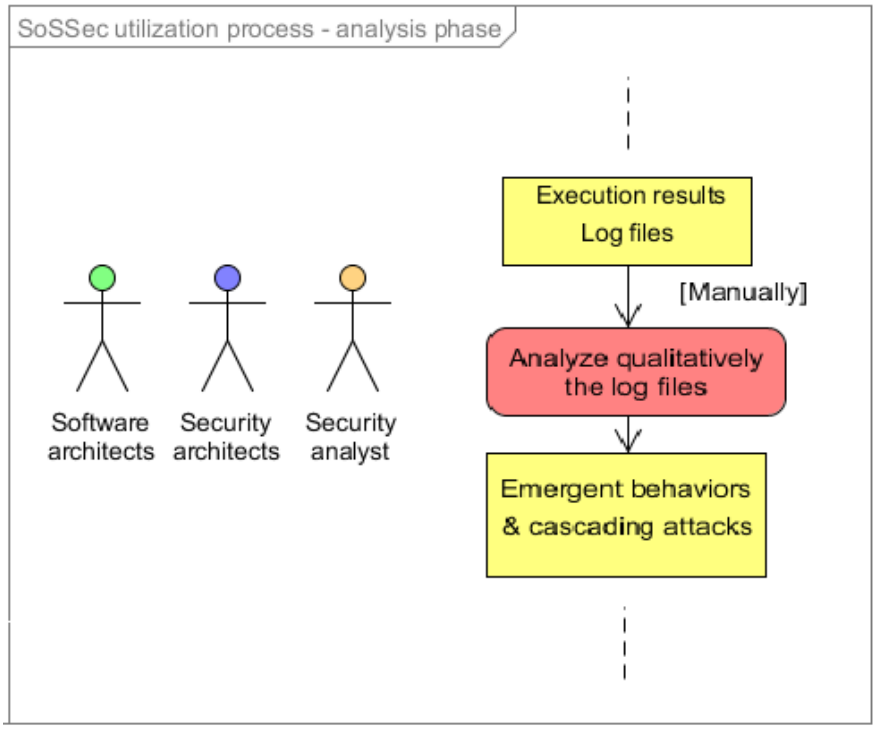


Fig. II.3.4 SoSSec Utilization Process - Analysis phase - Qualitative Analysis

triggered and connected through the CS/agent interactions and in consequence of validated pre-conditions/positive result of the vul_matching algorithm cf. subsection II.2.4.2).

After analyzing the log file, we can notice in figure II.3.5 the triggered vulnerabilities resulting from the AHMS agent/CS interactions. The cascade corresponding to the trigger of the three vulnerabilities outlined in red, as well as the matched pre/post-conditions in green and blue, as detailed in what follows.

Based on the structural architecture of our AHMS smart building SoS and its behavioral architecture described in the model and manifested in the execution results, and relying on the log files results revealing the emergent behaviors, we (the software and security architect and the security analyst) interpreted the AHMS secure architecture analysis results as follows:

1. Based on our scenario, **manageEnergy** is the initial operation, thus the pre-conditions *WeakPassword* and *AbsenceOfPassword* of its WeakAuthentication vulnerability are true by default. Therefore, the WeakAuthentication vulnerability *is_triggered* by default resulting in two post conditions *RemoteAccess* and *AccessPasswordInformation* (figure[2] II.3.6). These post-conditions are sent within all the outgoing messages from

[2]These figures have been manually processed with colors to better show the cascade between the triggered vulnerabilities

AHMSLogFile.txt

```
00:55:47.793 [BMS] INFO  src.generated.behaviours.automaticControlHVAC
              - Extends: [class jade.core.behaviours...]-- Initialising Behaviour
00:55:55.133 [Lighting] INFO  src.generated.behaviours.automaticControlLights
              - Extends: [class jade.core.behaviours...]-- Initialising Behaviour
00:56:02.715 [SmartGrid] INFO  src.generated.behaviours.manageEnergy
              - Extends: [class jade.core.behaviours...]-- Initialising Behaviour
...
-- Vulnerabilities - Vulnerability manageEnergy-Vul-WeakAuthentication (isTrigged: true)

00:56:02.825 [BMS] INFO  src.sec.Vulnerability  ------- automaticControlHVAC :
             Security Breached on automaticControlHVAC-Vul-DirectoryTraversal
             -- the pre-condition 'automaticControlHVAC-Post-AccessPasswordInformation' has been triggered.
            The post conditions which resulted in this breach are: manageEnergy-Post-AccessPasswordInformation
...
00:56:02.827 [BMS] INFO  src.sec.Vulnerability ------- automaticControlHVAC :
             Security Breached on automaticControlHVAC-Vul-PrivilageEscalation
             -- the pre-condition 'automaticControlHVAC-Post-RemoteAccess' has been triggered.
            The post conditions which resulted in this breach are:  manageEnergy-Post-RemoteAccess
...
-- Vulnerabilities --
    Vulnerability automaticControlHVAC-Vul-DirectoryTraversal (isTrigged: true)
    Vulnerability automaticControlHVAC-Vul-PrivilageEscalation (isTrigged: true)
...
    00:56:02.875 [Lighting] INFO  src.sec.Vulnerability  ------- automaticControlLights :
            Security Breached on automaticControlLights-Vul-CrossSiteScripting
            -- the pre-condition 'automaticControlLights-Post-ByPassAuthentication' has been triggered.
            The post conditions which resulted in this breach are: automaticControlHVAC-Post-ByPassAuthentication
...
-- Vulnerabilities - Vulnerability automaticControlLights-Vul-CrossSiteScripting (isTrigged: true)
...
```

Fig. II.3.5 AHMS smart building simulation results - Log file snippet analysis

this agent behavior/CS operation, thus, in our interaction scenario, they are sent to the **automaticControlHVAC**;

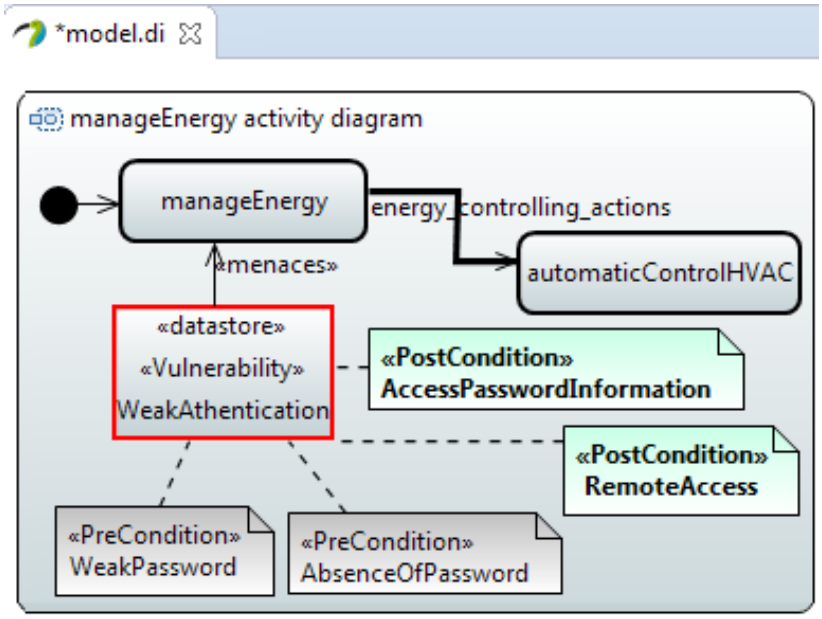


Fig. II.3.6 Smart Grid CS - triggered vulnerability on manageEnergy operation

2. At the reception, as detailed in the previous chapter subsection II.2.4.2, the *vul_matching* mechanism tests the matching between the received post-conditions and all the defined pre-conditions of the **automaticControlHVAC** vulnerabilities and once there is a match, the boolean variable *is_triggered* is set to "true" and a *security breached* event is logged. As we can see in figure II.3.7, the DirectoryTraversal vulnerability *is_triggered* because it has as pre-condition *AccessPasswordInformation* that matches the WeakAuthentication post-condition *AccessPasswordInformation*. In a similar way, the PrivilegeEscalation vulnerability *is_triggered* because it has as pre-condition *RemoteAccess* that matches the WeakAuthentication post-condition *RemoteAccess*;

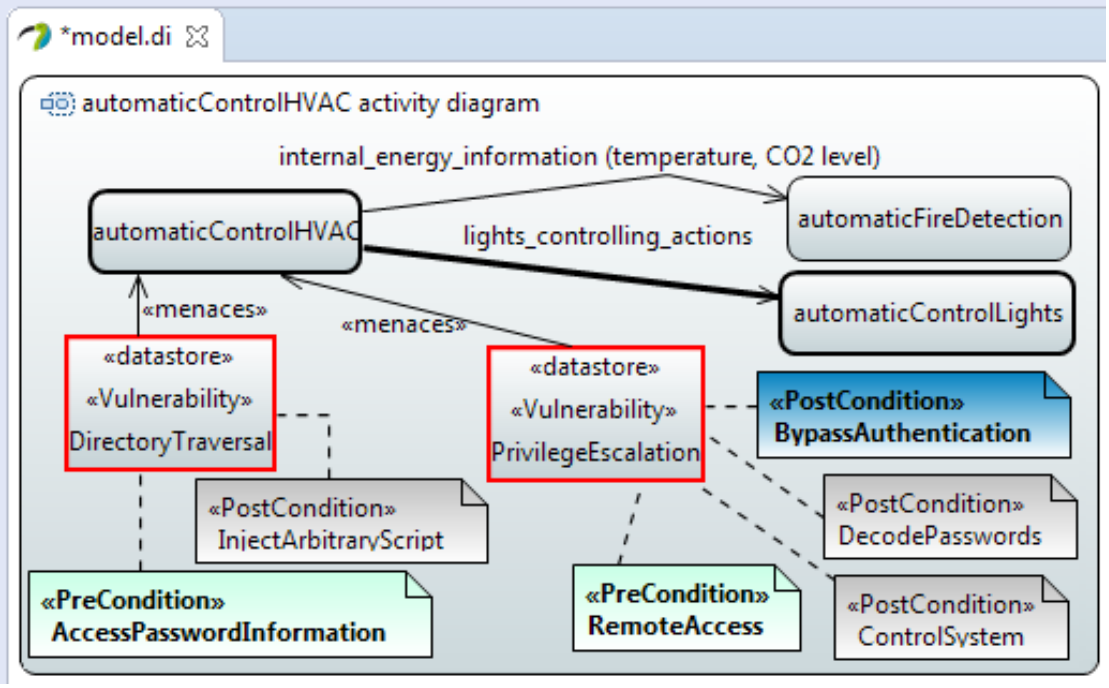


Fig. II.3.7 BMS CS - triggered vulnerability on automaticControlHVAC operation

3. In the same way, the CrossSiteScripting vulnerability of the *automaticControlLights* *is_triggered* due to the interaction between *automaticControlHVAC* and *automaticControlLights* CS operations/agent behaviors, engendering a matching between the ByPassAuthentication post- and pre-conditions (figure II.3.8).

From this analysis we can conclude that the designed architecture of the AHMS smart building is exposed to an emergent unknown cascading attack (figure II.3.9) resulting from the sequence of known triggered vulnerabilities (*WeakAuthentication, PrivilageEscalation* and *CrossSiteScripting*) due to the CSs interactions. In the same way, the other post-conditions *DecodePasswords* and *ControlSystem* of the triggered vulnerability *PrivilageEscalation*; and

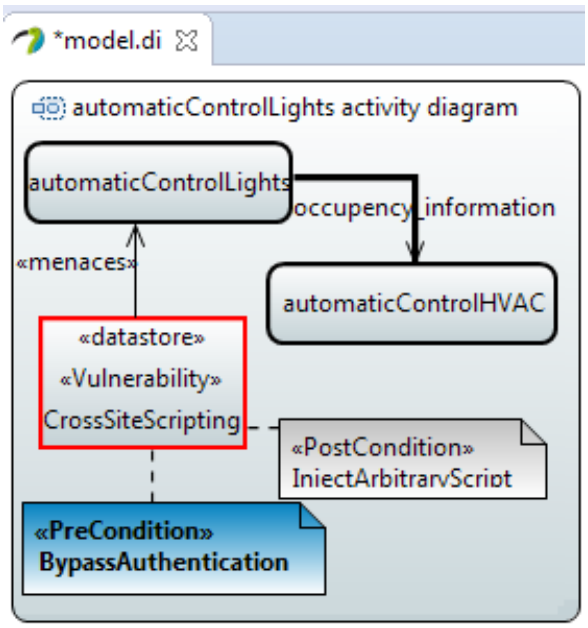


Fig. II.3.8 Lighting CS - triggered vulnerability on automaticControlLights operation

InjectArbitraryScript post-condition of the activated vulnerabilities *DirectoryTraversal* and *CrossSiteScripting* could match the post conditions of other pre-condition vulnerabilities forming several other possible cascading attacks.

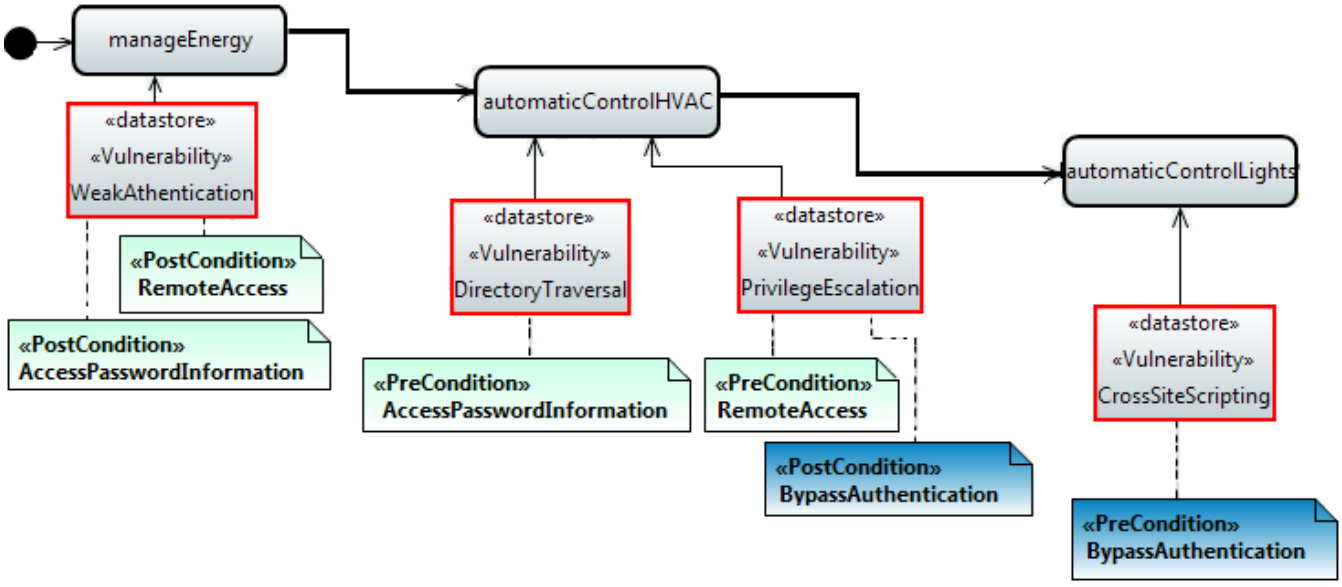


Fig. II.3.9 AHMS smart building cascading attack

II.3.2.4 Discussion of the AHMS secure architecture analysis results

The discovered cascading attack can lead to a *botnet*[3] opening the door to bigger attacks. Indeed, the *DirectoryTraversal* and *CrossSiteScripting* triggered vulnerabilities of the discovered cascading attack resulted in arbitrary script injections (*InjectArbitraryScript* postcondition), which represent software bots that could lead to a *botnet*. This *botnet* may serve to perform Distributed Denial-of-Service (DDoS) attack, steal data, send spam, and/or allow the attacker to access the device and its connection, affecting by that the smart building *integrity* (gained access or control of the critical automated CS by degrading or disrupting the CS operations through the injection of malwares) and *availability* (by preventing the delivery of the building CS functionalities). These latter security issues were the basis of the smart building attacks such as the Austrian hotel Ransomware attack, the Finnic central heating system DDoS, the USA Target store DDoS and the Google's Australian office hack, that we previously described in section I.3.2. Moreover, the cascading attack lead to the *BypassAuthentication* postcondition which allows the attackers to control the lighting CS and launch fake emergencies and alarms similar to the Google building attack; or to access the control management information in order to reveal personal data related to resident like their presence and absence, targeting by that the *confidentiality* of the AHMS smart building (unauthorized access to different CSs and their data, through unauthorized paths).

This discovery is therefore the main result that shows the utility of the SoSSec method in discovering the AHMS smart building security cascading attack emergent behaviors based on modeling and simulating its structural and behavioral architecture and its security vulnerabilities. This kind of reasoning and interpretation of the execution log files could be done by the SoS security analysis or automatically. Visualization techniques could help picturing the cascading attack, and a feedback could be sent to the software and security architects to propose/model a new architecture alternative that includes effective *security mechanisms* to prevent the discovered cascading attacks. To build their feedbacks, the architects may use pattern-based recommendations, machine learning methods, content-based filtering approaches or other techniques. These mechanisms could be incorporated in the design of the SoS architectures using SoSSecML and its graphical editor.

Furthermore, our method is conceived following the MDE mechanism, in a way that enables iterative modifications and an easy mapping between the modeling and the execution phases to analyze the adjusted secure SoS architecture (cf. utilization process II.2.7) until

[3] As defined by ISO/IEC 27032, a *botnet* is a collection of Internet-connected devices running malicious bots. A *bot* is a type of malware that run autonomously or automatically on a compromised devices and allows an attacker to take control over the affected devise. A botnet is used to perform malicious activities under remote direction.

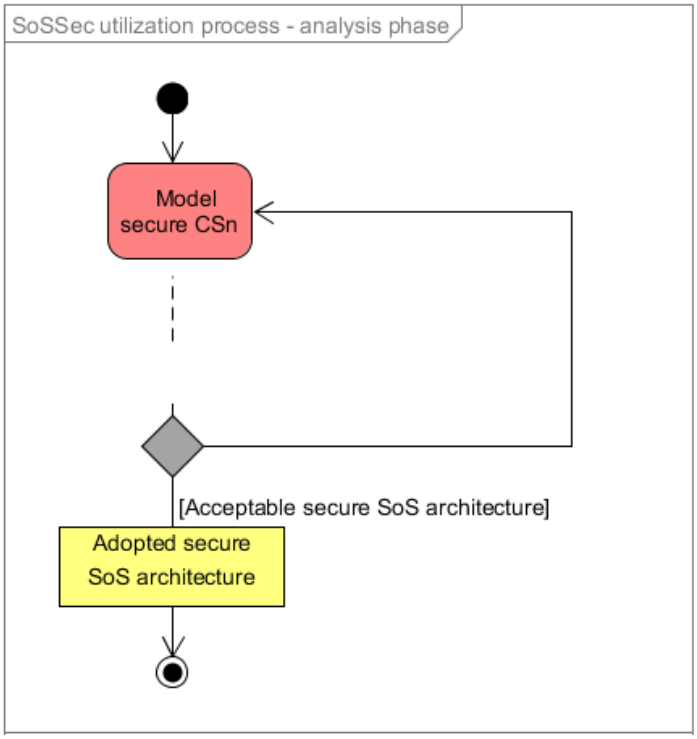


Fig. II.3.10 SoSSec Utilization Process - Analysis phase - Iterative aspect

reaching an acceptable level of security, as shown in the snippet II.3.10. Indeed, the software architect, the security architect, and security analysts decide whether the secure SoS architecture is acceptable in terms of compliance to the needed security level, or there is a need to modify it and/or define some security mechanisms and go through the whole process again.

The results of the AHMS case study were discussed and analyzed in a meeting with the AHMS team (cf. AHMS case study protocol section I.3.3 - figure I.3.4) which approved the pertinence of the modeled scenario and confirmed the utility of the results in improving the security of the current AHMS building, as well as the prominent potential of the SoSSec method to improve the modeling and analysis of the secure software architecture of the future smart buildings of the Adelaide university.

II.3.3 Case study limitations

Our SoSSec method addressing *Systems-of-Systems modeling* and *analysis* while also considering *security* concerns is an innovative method as there is only a very limited number of primary studies, recently published in 2016 and 2017, that address similar challenges (cf. section II.1.1).

By applying our SoSSec method (SoSSecML language, tools and utilization process), on the AHMS smart building case study, we showed the operability of the method and its potential in answering our research questions. However, it is imperative to mention that our case study has certain limitations related to validity issues.

Firstly, the case study data was collected from several meetings with a reduced number of specialists. We did not discuss the profound details with specific experts representing the different organizations in charge of the various AHMS constituent systems. In addition, even if we have a good expertize in the smart building and their security domains, there could be some additional vulnerabilities that we couldn't catch when defining the list of possible vulnerabilities for our case study CSs, therefore we didn't model them.

Secondly, by arguing that a smart building, in particular the AHMS, is an SoS with respect to the OMGEE specific characteristics, and by applying the SoSSec method on the AHMS without any need of refining it, we assume that it could be generalized to other application domains without complications. However, this case study was realized in a two months duration in the context of this Ph.D., accordingly, due to time constraints and in some parts confidentiality constraints, the scenario selected for our AHMS case study remains relatively simple compared to SoS interaction scenarios. Therefore, an additional work should be conducted to ensure the scalability of our SoSSec method. In the same context, an additional work is needed to study the capacity of SoSSec in scaling out by applying it to bigger case studies with extensive interaction scenarios such as smart cities.

Thirdly, the application of SoSSec by following its detailed utilization process and using its tools, shows the usability of our method in modeling and analyzing a secure SoS architecture. The different users who used the method in the context of the AHMS case study (notably Mme. EL HACHEM and M. AL KHALIL) attested the relatively ease of use of the method. In addition, the easiness of the SoSSec usability was testified by three additional master students who used the method in the context of their research projects/internships. However, an extended analysis of the usability of our SoSSec method should be realized to ensure its ease of use.

Finally, the case study and its results were validated with the AHMS team and by few additional users (four master students), but it was not confronted with security experts. Therefore, the SoSSec evaluation needs to be further enforced by validating the results with security experts.

However, the case study has been conducted following a methodical protocol, which allowed capturing important informations, these limitations included, and which allowed and will allow the improvement of our SoSSec method.

II.3.4 Conclusion

In this chapter, we described in details how we applied the analysis phase of our SoSSec method, following its utilization process, to the AHMS smart building case study. The use of the SoSSec generator, the extended MAS execution platform and the analysis mechanism shows their operability to execute the AHMS smart building secure architecture and analyze the results to discover the related emergent cascading attacks, and by that to achieve our objectives for this second part (cf. Part II Introduction).

The analysis results of one possible interaction scenario show that the AHMS smart building secure architecture is exposed to a security cascading attack giving rise to high impact security issues such as DDoS, fake alarms and emergency situations. In fact, having small vulnerabilities remaining in the CSs of the AHMS and exploited at the SoS execution makes this SoS exposed to a cascading attack with significant effects.

By modeling and analyzing the secure architecture of the most recent smart building of the university of Adelaide (the AHMS building) we answered the first research question that we defined with the AHMS team in our case study protocol (cf. AHMS case study protocol section I.3.3).

Discovering such attacks at the architecture stage of SoS development life cycle enables their resolution at this early phase of development, and thus save cost and development time, and protect the SoS from high impact cyber attacks and massive resulting damages. For the AHMS case study, the SoSSec team suggested possible requirements to improve the AHMS smart building security (cf. section I.3.3 figure I.3.4). The AHMS team founded that the resulting output valuable and the SoSSec method very interesting for the modeling and analysis of the University of Adelaide future buildings (precisely those that will be built in the context of the smart campus master plan 2016-2035 project, cf. I.3) and improving their security. These results answered the second research question in our case study protocol (cf. AHMS case study protocol section I.3.3).

Part II Conclusion

In this second part, we undertook the secure Systems-of-Systems (SoS) analysis challenge. We examined Multi-Agent Systems (MAS) and we proposed a semantic alignment between the SoS and MAS domains. In addition, we proposed a MAS MetaModel security extension and we used the Model-Driven Engineering (MDE) approach to define the semantic mapping between the described alignments. Moreover, we implemented the MAS MetaModel extensions in a MAS platform that we used to execute and analyze the modeled secure SoS architecture and discover the cascading attacks to which this architecture is exposed, answering by that our second research question (RQ2).

We analyzed the existing approaches for SoS architecture execution and/or analysis and we identified their potentials and limitations. Our study revealed that: 1) none of the related-work approaches fully address our main objective, that of analyzing secure SoS architectures to discover the security emergent behavior; 2) dynamic analysis approaches are an appropriate technique to analyze the SoS architecture mainly due to the fact that dynamic simulation implements the fundamental dynamics associated to the CS interactions to help identifying the key behaviors that influence the overall SoS; 3) MAS simulation seems to be the most suited and actively used analysis technique to express the SoS aspects; 4) there is a need for well-defined relations/alignments between the concepts of the secure SoS architectures modeled using SoSSecML and the semantics expressed by the corresponding MAS MetaModel concepts. These alignments should also express the semantics of the SoSSecML security concepts such as vulnerabilities, pre- and postconditions and attacks; 5) the MAS platform adopted for the secure SoS architecture execution should be extended to include the MAS MetaModel extensions.

To address the identified needs, we presented the second phase of our SoSSec method. We argued the usefulness of MAS for SoS simulation/execution due to the similarities between MAS and SoS concepts. Moreover, we extended MAS with security concepts to express the semantics of all the secure SoS architecture concepts defined by our SoSSecML language, offering by that an unambiguous MAS simulation approach for secure SoS architectures. This approach could be re-used and further extended to model and analyze other security

properties and concerns such as security mechanism and SoS security goals. Furthermore, we defined a semantic mapping generator, a set of Model-To-Text (MTT) transformation rules, from the concepts of our SoSSec Modeling Language (SoSSecML MetaModel) to its semantic elements expressed by the extended MAS MetaModel concepts. In addition, we customized a proper logging mechanism to record the execution results in order to analyze them and extract the information related to the unknown security cascading attack emergent behavior. Then, we completed the SoSSec utilization process by an analysis phase that guides the use of the SoSSec generator (MTT rules) and the execution platform.

Finally, we illustrated our contribution by analyzing the secure SoS architecture of the AHMS real-world smart building SoS that we modeled in the first part of this thesis. Afterwards, we used the SoSSec generator to map the AHMS model to the extended JADE MAS simulation platform, where we executed the AHMS model and we logged the results. We analyzed the execution results to discover the possible cascading attacks related to the modeled AHMS interaction scenario. The results confirm that our method discovers cascading attacks that arise from a succession of vulnerabilities throughout the AHMS CS interactions. The current AHMS architecture is exposed to security issues that may serve an attacker to perform Distributed Denial-of-Service (DDoS) attacks, steal data, send spam, launch fake emergencies and alarms and/or access the control management information in order to reveal personal data related to residents. These security issues target the AHMS confidentiality, availability and integrity, affecting by that the whole AHMS security.

SoSSec allows the discovery of unknown emergent sequence of known triggered vulnerabilities (leading to the previously mentioned emergent cascading attacks) early at the architecture phase of the SoS development life cycle. Furthermore, the SoSSec iterative aspect allows the analysis of a new modified/corrected secure SoS architecture through the easy mapping between the modeling and analysis phases. Therefore, SoSSec helps improving the architecture and preventing the massive damages of the cyber attacks targeting SoS functionalities and impacting people's safety and security, all that while avoiding time and cost wastage of later development changes.

Contribution: An extension of a Multi-Agent System (MAS) MetaModel and platform (JADE extension), a model-driven based generator (set of Model-To-Text transformation rules) and a vulnerability matching mechanism, along with the corresponding processes (generator and MAS extension construction process and SoSSec utilization process for the analysis phase) to simulate and analyze SoS models designed using our Domain Specific Language (SoSSecML) to discover possible unknown successions/sequences of security vulnerabilities leading to cascading attacks (security emergent behaviors), and its application on a real-world AHMS smart building case study.

General Conclusion

The proliferation of Systems-of-Systems (SoS) in the last years make it an important research field with a wide application in many domains such as smart cities, modern transportation systems, emergency management systems, health-care and smart buildings. SoS are usually defined as Systems composed of distributed, independent Constituent Systems (CSs) that interact to accomplish a global goal that none of the CSs can achieve individually. SoS are identified with their specific characteristics such as operational and managerial independence of the CSs, geographic distribution, emergent behavior and evolutionary development. SoS specific characteristics and complexity arising from the uncertainty of behavior generate many engineering challenges such as developing methodologies with languages, tools and processes to conceptually describe, simulate and analyze SoS architectures. Moreover, SoS characteristics and complexity embody additional challenges related to the SoS non-functional properties, in particular security. One of the main SoS security concerns is the cascading attacks. These attacks usually start from single known vulnerabilities initially judged as insignificant or low-impact for the CS on which they were identified. These vulnerabilities can be triggered and connected through the interactions between the CSs (interactions needed for the achievement of the SoS global goal). The unknown cascades/sequences of several triggered vulnerabilities lead to security cascading attacks with high-impact in the context of SoS.

Fulfilling the objectives and contributions

In this thesis, we addressed SoS *modeling* and *analysis* challenges, while considering the SoS specific characteristics and the cascading attacks security emergent behavior. We proposed a method called Systems-of-Systems Security (SoSSec) to overcome these challenges early at the *architecture phase* of the SoS development life cycle 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. We followed the guidelines of Model Driven Engineering (MDE) to develop our SoSSec method encompassing a modeling language, modeling and

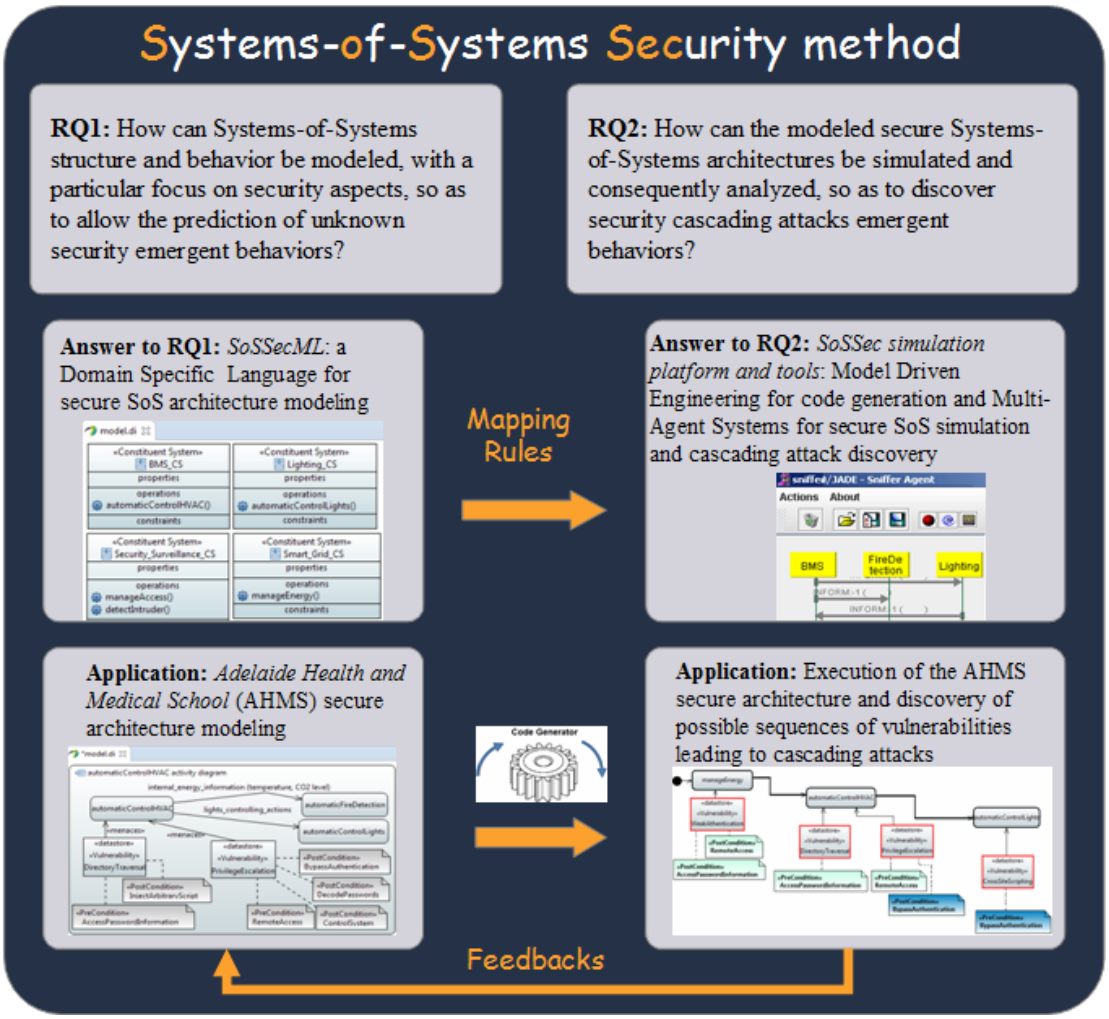


Fig. II.3.11 SoSSec method for secure SoS modeling and analysis

simulation tools together with their utilization process describing how the language and tools should be used (figure II.3.11). To illustrate our proposed method in terms of modeling and simulating secure SoS architecture, and discovering attacks, we conducted a case study on a real-life smart building SoS, the Adelaide University Health and Medical School (AHMS).

SoSSec modeling contributions:

More particularly, in the modeling part, to answer our first Research Question RQ1 "How MDE approaches could be extended and used to model SoS structure and behavior with a particular focus on security aspects to allow the prediction of unknown security emergent behaviors?", **we proposed a Domain Specific Language (DSL) called SoSSecML** (SoSSec Modeling Language). SoSSecML extends the System Modeling Language (SysML) Meta-

Model with structural and behavioral concepts and relationships specialized for SoS such as constituent system, operation and organization, and others particular for security cascading attacks such as vulnerability, pre-conditions, post-conditions and security mechanism. The extensions are implemented as a profile using MDE mechanisms and tools. **SoSSecML has many strong points**:

- Built upon SysML which is, based on our related work study, the most suited and actively used and/or extended language to model SoS architecture, specifically CSs and their interfaces;
- Allows *secure SoS architecture* modeling taking into account their specific characteristics and security aspects, as follows:
  - SoSSec handles the *managerial* and *operational independence* characteristics of an SoS by modeling separately the structure of the CSs, the organization to which they belong as well as their interfaces and interactions;
  - It also respects the *geographic distribution* characteristic since MAS allows the distributed simulation for the execution of the models (as detailed in the analysis part);
  - SoSSec addresses the *emergent behavior* of SoS through the modeling of vulnerability of each CS, and analysis and prediction of the cascading attack security emergent behavior;
  - SoSSec partially covers the *evolutionary development* of the SoS since it is an iterative process that allows the modeling and analysis of several architecture alternatives each time a CS is added, modified or deleted.
- Offers a rich representation of the secure SoS, which subsequently allows, at the simulation and analysis phase, some types of emergent behavior to manifest and therefore become analyzable, in particular the unknown sequences of triggered vulnerabilities resulting in emergent security cascading attacks;
- Extensible, portable and customizable DSL due to the fact that it is developed using MDE mechanisms (Meta-modeling, profile and semi-automatic transformation mechanisms);

Along with SoSSecML, **we proposed and implemented the corresponding graphical editor** (visual modeling tool) that extends the Papyrus Eclipse-based and open source tool. We benefited from MDE transformation mechanisms to automatically generate our tool offering by that a tool with the following **strengths**:

- A tool that extends the Papyrus open source Eclipse-based tool, actively used by the software engineering community and properly documented and maintained by the CEA commission and supported by active forums, which may be an important factor in further acceptation and adoption in the industry;
- Not only a model-based tool adapted for secure SoS architectures specific domain, but also a *customizable*, *reusable* and *extensible* tool that could be adapted for other related modeling domains to suit the architect needs (for example a tool adapted for secure or safe smart cities specific domain) by extending or filtering the existing palette or by integrating new/existing Eclipse plug-ins;
- Platform independent modeling paradigm to design a secure SoS architecture regardless of the specific technological platform that will be used to execute it.

In addition, **we defined an utilization process** to provide guidance on the use of SoSSecML and its tool. This main advantage of this process is to guide any user having basic knowledge on the SoS application domain, to use the SoSSec language and tools to model and analyze the secure architecture of the SoS. The process describes step by step, the needed actors and tools for each modeling and analysis activity. We followed this utilization process to model a real-life SoS as presented in the next paragraph.

Moreover, **we conducted a case study on a real-case smart building SoS**, the AHMS building. We followed the SoSSec process to apply the SoSSec DSL, tools and utilization process on the case study. We modeled the building structural and behavioral aspects, and some of their vulnerabilities together with their pre- and post- conditions (extracted from vulnerability knowledge bases like CVE). **Results show**:

- The ability of the proposed language in modeling the smart building as an SoS composed of several CSs such as the Heating, Ventilation and Air-Conditioning (HVAC) CS, the smart grid CS and the lighting CS, taking into account the OMGEE[4] specific characteristics and the cascading attack security elements, as testified by the SoSSec research team (who applied the method to model the AHMS smart building) and confirmed by the AHMS team (who offered the needed data as input for the case study and evaluated the outcome of this study and its usefulness for modeling the future smart buildings of the Adelaide University);
- The relatively ease of use of the proposed method (SoSSecML and its graphical editor) by following the detailed guidance (actors, activities and tools) of the defined utilization

[4]OMGEE: An acronym referring to the five SoS specific characteristics defined by Maier[75]: Operational independence of the CSs, Managerial independence of the CSs, Geographic distribution, Emergent behavior and Evolutionary development.

process, as experienced by the few tool users which had limited expertises in the SoS and MDE domains;

- The increased probability of adoption by other users, as the language and tool are accompanied by guidance and activity description documentation, showing their intended use;
- The iterative aspect of our SoSSec method, allowing a continual refinement of the secure smart building architecture.

SoSSec analysis contributions:

For the analysis part, to answer our second Research Question RQ2 "How can the modeled secure Systems-of-Systems architectures be simulated and consequently analyzed, so as to discover security cascading attacks emergent behaviors?", **our proposed method SoSSec encompasses an extension of Multi-Agent Systems with security concepts** to execute secure SoS architectures and discover and analyze the possible unknown sequence of vulnerabilities leading to emergent security cascading attacks. This proposition has many **strong points**:

- A dynamic simulation approach for SoS architecture execution and analysis mainly to handle the emergent behaviors resulting from CS interactions, as well as to support the early discovery and anticipation of unforeseen (unknown known) security issues that need to be corrected at the architectural phase of the SoS development life cycle;
- A Multi-Agent Systems (MAS) based simulation allowing an expressive, unambiguous and useful execution of the SoS architecture due to the similarities between SoS and MAS concepts such as local views, decentralization, self organization, large problem solving and legacy systems leveraging;
- An extension of the MAS MetaModel with security concepts and a matching mechanism to allow the simulation and analysis of secure SoS architectures and to support the early discovery and anticipation of unforeseen security issues;
- Avoiding the combinatorial explosion of existing attacks analysis approaches that results from the analysis of all possible attack paths. Indeed, given a SoS global goal(s), the CSs of this SoS are supposed to interact, to use their global functionalities/operations defined on their interfaces, to accomplish this/these global goal(s). In this context, the SoSSec matching mechanism tests if a vulnerability should be triggered or not, only for the CS interactions needed for the realization of the SoS

global goal(s). In consequence, our SoSSec mechanism allows the discovery of the possible sequence of triggered vulnerabilities resulting only from the execution of CSs operations;

- An iterative method allowing 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. Addressing the SoS security issues early at the architectural phase of the development life cycle help saving cost, development time, and protecting the SoS from high impact attacks.

We implemented the proposed MAS extension by extending the JADE platform. In addition, we followed the MDE approach and used its mechanisms to **define a model driven based generator** - a set of Model-To-Text transformation rules - to (semi)automatically map the secure SoS architecture to executable artifacts. We defined these rules using the eclipse-based Acceleo open source tool. The **advantages** of our generator and execution/simulation and analysis platform are the following:

- Automatic mapping (MTT rules) from the secure SoS architectures to their execution in MAS, which enable easy extension/evolution of our method and its tools;
- The effectiveness of MAS concepts in expressing the semantics of the AHMS architectural elements, and therefore to properly execute the AHMS architecture and accurately capture the AHMS CS interactions;
- A simulation platform compatible with the MAS standard (FIPA), based on the popular JADE open source platform which is well documented, widely used, regularly updated and actively maintained;
- A custom log file issued from the execution/simulation of the secure SoS architecture and capturing and revealing the 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);

In addition, **we complemented our utilization process** to describe the actors and guide the use of the SoSSec generator, simulation platform and analysis mechanism.

Afterwards, we followed the utilization process to **apply our SoSSec method** to generate the corresponding code into the extended MAS platform. The generated code was executed **to simulate the AHMS secure architecture** - designed using SoSSecML - and **analyze the**

results to discover the unknown possible sequence of triggered vulnerabilities leading to emergent cascading attacks. The obtained **results show**:

- The ability of the method in performing an easy mapping from the AHMS secure architecture to expressive MAS executable notations, by following the guidance of our detailed utilization process;

- The ability of the SoSSec method in discovering and analyzing one of the specific SoS security issues: the unknown cascading attacks. Indeed, given the AHMS SoS with several CSs that interact to achieve a common/global goal (enhance the automation and increase energy savings). Each independent CS have a list of known unresolved vulnerabilities. When the CSs interact to achieve the AHMS global goal , the CS vulnerabilities are connected/concatenated in an unknown way. This succession of vulnerabilities give rise to the cascading attacks. Therefore, the use of MAS simulation to execute the AHMS secure SoS architectures helped expressing the AHMS CSs interactions as described in the AHMS architectures. Moreover, the matching mechanism that we defined and implemented, leads to prominent results in terms of discovery of possible sequence of vulnerabilities in the AHMS architecture that may lead to high impact attacks such as Distributed Denial of Service attacks, fake alarms and emergency situations. The importance of this mechanism resides in the discovery of the attacks arising only from the triggered vulnerabilities related to the executed operations, avoiding by that the combinatorial explosion. These results were conformed by the AHMS team who found the resulting output valuable and the SoSSec method very interesting for the modeling and analysis of the University of Adelaide future smart buildings to improve their security.

As shown, the contributions proposed have several strong points, and answers the objectives of this thesis by answering to the defined research questions. However, it is imperative to mention that our method has certain limitations that could be improved as discussed in the following section.

Perspectives

Based on our work, we present in this section many perspectives and a number of possible future directions that would advance our research. Some of these perspectives are motivated by some ongoing work and issues that our study has raised. The others are defined with the intention of extending our SoSSec method to address various SoS modeling and analysis challenges and numerous SoS non-functional properties.

Improve the SoSSec method for secure SoS architecture modeling and analysis:

In this work we modeled the CSs as black boxes with several operations on their interfaces, without modeling their internal security issues arising from the vulnerabilities of their internal components. It is maybe worth to model the detailed secure architecture of each CS in addition to its interface and global interactions, and then execute this architecture and analyze the impact of the internal security issues on the SoS security. In addition, it is even feasible to use our SoSSec method to model and analyze secure monolithic complex systems. Specially that the SoSSec method allows this activity, since it extends the SysML language and therefore it permits the modeling of the details of each system.

Moreover, some improvements could be envisaged to enhance the secure SoS architecture analysis 1) by the aggregation of scores for quantitative attack evaluation by quantifying the security impact of the cascading attacks. The scores can be calculated by assigning Common Vulnerability Scoring System (CVSS) metrics to the vulnerabilities extracted from the Common Vulnerabilities Exposure (CVE); 2) by visualizing the simulation results (log files traces) specially the cascading attacks and using data analysis techniques [59][51] to automatically track-back vulnerabilities, in order to give a visual and more meaningful feedback to the architects; 3) by employing security design patterns [65] to support the architects to implement corrective strategies and solutions such as security mechanisms and 4) by recursively analyzing the SoS architecture alternatives after including and corrective actions, until reaching an acceptable security level.

Consistently we intend to update the modeling and analysis tools to meet the SoSSec method improvements. Furthermore, we intend to apply SoSSec method (process, modeling language and tools) on several SoS application domains to investigate its usability, completeness and finally validated it, by industry software and security architects, as well as by novice student users. In addition, seeing that when working with SoS, architects have to deal with a whole new level of abstraction/modeling, maybe it would be interesting to investigate how this need of "higher" abstraction impacts the architects and how do they deal with it when using SoSSec.

Develop the SoSSec method to address the various categories of *emergent behaviors*:

In this work we addressed the *unknown knowns emergent behavior* that could be explained in the context of SoS by the fact that each CS possesses a list of its own vulnerabilities (*known*), but since there is no central authority in the SoS, this list of vulnerabilities will remain *unknown* by other CS, hence the nomination *unknown known vulnerabilities*. However, there are three other types of emergent behaviors identified in [113], in particular: Known knowns, known unknowns and unknown unknowns. In our future work, we intend to investigate the extension of SoSSec to cover the evoked emergent behaviors.

Moreover, we plan to extend the SoSSec method 1) to conceive dynamic architectures for adaptive SoS that should evolve with the continuous SoS changes and uncertainties, in order to properly address the *evolutionary development* specific SoS characteristic; 2) to consider the interactions between the SoS and its environment and their impact on the cascading attack prediction. More precisely, we have in mind to extend the SoSSec MetaModel to include context/environment concepts and corresponding relationships, and we will take advantage of the MAS semantic agent knowledge/belief base and environment notations to express the semantics of the added concepts.

Create links with the previous and posterior phases of the SoS development life cycle:

We aim to investigate how the modeling phase of the SoSSec method could be linked to the requirement phase of the SoS development life cycle, in order to define and model the SoS security requirements. This link permits to consider the SoS security goals in a way that helps refining these goals when using SoSSec for the secure SoS architecture. Moreover, we intend to verify and validate the SoS architectures conceived using the SoSSec method by using one of the well-known formal methods such as model checking to test the validity and check the correctness of the SoS architecture and ensure that the expected behavior of the modeled SoS is properly represented.

Extend SoSSec to include other non-functional properties:

In this thesis, we focused on security. However, an interesting investigation is the expansion of the SoSSec method to model and analyze other Non-Functional Properties (NFP) such as safety, trust and privacy together with trade-off analysis methods to compare the architectures in terms of several NFP such as the interplay between security and safety or privacy, trust, security and performance.

To handle these SoS modeling and analysis challenges, several Computer Science domains could be leveraged and combined. To complement SoS specifications, modeling and

analysis, Business Process Modeling/Information Systems seems to be promising. To study the negative effect of uncertainty - coming from system hazard or environmental conditions - on the SoS objectives, Risk analysis methods could be a promising. To design self-organizing and adaptable architectures, Artificial intelligence seems promising. To identify emergent behaviors by collecting and discovering knowledge from diverse, distributed, and heterogeneous constituents, Big Data and machine learning techniques seem auspicious.

Last but not least, there are still numerous challenges and open-ended issues to SoS engineering that must be addressed before this fulfills all its great potential. We hope that the contributions of this thesis and the envisaged future work offer a step closer towards this end.

References

[1] *INCOSE Systems Engineering Body of Knowledge, version 1.6*. INCOSE UMS, March 2016. URL http://www.bkcase.org/sebok/.

[2] R. Abercrombie and F. Sheldon. Security analysis of smart grid cyber physical infrastructures using game theoretic simulation. In *IEEE Symposium Series on Computational Intelligence*, pages 455–462, Dec 2015. doi: 10.1109/SSCI.2015.74.

[3] R. Alaguvelu, D. Curry, and C. Dagli. Fuzzy genetic algorithm approach to generate an optimal meta-architecture for a smart, safe efficient city transportation system of systems. In *11th System of Systems Engineering Conference (SoSE)*, pages 1–6, June 2016. doi: 10.1109/SYSOSE.2016.7542935.

[4] B. Ali, U. Manzoor, and B. Zafar. eJADE-S: Encrypted jade-s for securing multi-agent applications. *The World Congress in Computer Science, WORLDCOMP2015*, 2015. URL http://worldcomp-proceedings.com/proc/p2015/ICA3062.pdf.

[5] L. Apvrille, L. Li, and Y. Roudier. Model-driven engineering for designing safe and secure embedded systems. In *Architecture-Centric Virtual Integration (ACVI)*, pages 4–7, April 2016. doi: 10.1109/ACVI.2016.6.

[6] A. Avizienis, J. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. Dependable Secure Computing*, 1(1):11–33, January 2004. ISSN 1545-5971. doi: 10.1109/TDSC.2004.2. URL http://dx.doi.org/10.1109/TDSC.2004.2.

[7] J. Axelsson. A systematic mapping of the research literature on system-of-systems engineering. In *10th System of Systems Engineering Conference (SoSE)*, pages 18–23, May 2015. doi: 10.1109/SYSOSE.2015.7151918.

[8] A. Babar. Smart cities: Socio-technical innovation for empowering citizens. *Australian Quarterly*, 87(3):18 – 25, 2016.

[9] P. Baker, S. Loh, and F. Weil. Model-driven engineering in a large industrial context — motorola case study. In *Proceedings of the 8th International Conference on Model Driven Engineering Languages and Systems*, MoDELS'05, pages 476–491, 2005. ISBN 3-540-29010-9, 978-3-540-29010-0. doi: 10.1007/11557432_36. URL http://dx.doi.org/10.1007/11557432_36.

[10] W. Baldwin, B. Sauser, and R. Cloutier. Simulation approaches for system of systems: Events-based versus agent based modeling. *Procedia Computer Science*, 44:363 – 372, 2015. ISSN 1877-0509. doi: http://dx.doi.org/10.1016/j.procs.2015.03.032. URL

http://www.sciencedirect.com/science/article/pii/S1877050915002689. Conference on Systems Engineering Research.

[11] British Broadcasting Corporation BBC news. Tomorrow's buildings: Help! my building has been hacked. 2016. URL http://www.bbc.com/news/technology-35746649.

[12] F. Bellifemine, G. Caire, and D. Greenwood. *Developing Multi-Agent Systems with Jade*. John Wiley & Sons, 2007. ISBN 978-0-470-05747-6.

[13] G. Beydoun and G. Low. Generic modelling of security awareness in agent based systems. *Information Sciences*, 239:62 – 71, 2013. ISSN 0020-0255. doi: http://dx.doi.org/10.1016/j.ins.2013.03.039. URL http://www.sciencedirect.com/science/article/pii/S0020025513002442.

[14] G. Beydoun, G. Low, H. Mouratidis, and B. Henderson-Sellers. A security-aware metamodel for multi-agent systems. *Information and Software Technology*, 51(5):832 – 845, 2009. ISSN 0950-5849. doi: http://dx.doi.org/10.1016/j.infsof.2008.05.003. URL http://www.sciencedirect.com/science/article/pii/S0950584908000724. SPECIAL ISSUE: Model-Driven Development for Secure Information Systems.

[15] J. Bézivin. In search of a basic principle for model driven engineering. *Novatica Journal, Special Issue*, 5(2):21–24, 2004.

[16] S. Bijani and D. Robertson. A review of attacks and security approaches in open multi-agent systems. *Artificial Intelligence Review*, 42(4):607–636, Dec 2014. ISSN 1573-7462. doi: 10.1007/s10462-012-9343-1. URL https://doi.org/10.1007/s10462-012-9343-1.

[17] J. Boardman and B. Sauser. System of systems - the meaning of of. In *IEEE/SMC International Conference on System of Systems Engineering*, pages 6 pp.–, April 2006. doi: 10.1109/SYSOSE.2006.1652284.

[18] 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. volume 30, pages 33–44, 2006. URL http://www.informatica.si/index.php/informatica/article/view/71.

[19] M. Brambilla, J. Cabot, and M. Wimmer. *Model-Driven Software Engineering in Practice*. Morgan & Claypool Publishers, 1st edition, 2012. ISBN 1608458822, 9781608458820.

[20] P. Brereton, B. Kitchenham, D. Budgen, and W. Li. Using a protocol template for case study planning. In *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering*, EASE, pages 41–48, 2008. URL http://dl.acm.org/citation.cfm?id=2227115.2227120.

[21] J. Bryans, J. Fitzgerald, R. Payne, A. Miyazawa, and K. Kristensen. Sysml contracts for systems of systems. In *9th International Conference on System of Systems Engineering (SOSE)*, pages 73–78, June 2014. doi: 10.1109/SYSOSE.2014.6892466.

[22] E. Cavalcante, A. Medeiros, and T. Batista. Describing cloud applications architectures. In *Proceedings of the 7th European Conference on Software Architecture*, ECSA'13, pages 320–323, 2013. ISBN 978-3-642-39030-2. doi: 10.1007/978-3-642-39031-9_29. URL http://dx.doi.org/10.1007/978-3-642-39031-9_29.

[23] R. Cavalcante, I. Bittencourt, A. Silva, M. Silva, E. Costa, and R. Santos. A survey of security in multi-agent systems. *Expert Systems with Applications*, 39(5):4835 – 4846, 2012. ISSN 0957-4174. doi: http://dx.doi.org/10.1016/j.eswa.2011.09.130. URL http://www.sciencedirect.com/science/article/pii/S0957417411014539.

[24] L. Caviglione, J. Lalande, W. Mazurczyk, and S. Wendzel. Analysis of Human Awareness of Security and Privacy Threats in Smart Environments. In *3rd International Conference on Human Aspects of Information Security, Privacy and Trust*, Los Angeles, United States, August 2015. doi: 10.1007/978-3-319-20376-8. URL https://hal.inria.fr/hal-01182303.

[25] Cupertino Electric Incorporation CEI. Cyber security in smart buildings part 1: Disruptive attacks. 2015. URL http://www.cei.com/about-cei/media-room/blog/cyber-security-in-smart-buildings-part-1-disruptive-attacks.

[26] M. Challenger, G. Kardas, and B. Tekinerdogan. A systematic approach to evaluating domain-specific modeling language environments for multi-agent systems. *Software Quality Journal*, 24(3):755–795, 2016. ISSN 0963-9314. doi: 10.1007/s11219-015-9291-5. URL http://dx.doi.org/10.1007/s11219-015-9291-5.

[27] V. CHIPRIANOV. *Collaborative Construction of Telecommunications Services. An Enterprise Architecture and Model Driven Engineering Method.* Theses, Télécom Bretagne, Université de Bretagne-Sud, January 2012. URL https://tel.archives-ouvertes.fr/tel-00719634.

[28] 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 (CIEL'2014)*, pages 137–151, Paris, France, June 2014. URL https://hal.archives-ouvertes.fr/hal-01082097.

[29] J. Dahmann. System of systems pain points. *INCOSE International Symposium*, 24(1):108–121, 2014. ISSN 2334-5837. doi: 10.1002/j.2334-5837.2014.tb03138.x. URL http://dx.doi.org/10.1002/j.2334-5837.2014.tb03138.x.

[30] J. Dahmann. The state of systems of systems engineering knowledge sources. In *10th System of Systems Engineering Conference (SoSE)*, pages 475–479, May 2015. doi: 10.1109/SYSOSE.2015.7151979.

[31] J. Dahmann and G. Roedler. Moving towards standardization for system of systems engineering. In *11th System of Systems Engineering Conference (SoSE)*, pages 1–6, June 2016. doi: 10.1109/SYSOSE.2016.7542953.

[32] J. Dahmann, G. Rebovich, M. McEvilley, and G. Turner. Security engineering in a system of systems environment. In *IEEE International Systems Conference (SysCon)*, pages 364–369, April 2013. doi: 10.1109/SysCon.2013.6549907.

[33] D. David, G. Roedler ESEP, and K. Forsberg, editors. *INCOSE Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities, 4th Edition*, page 678–686. John Wiley and Sons, 2015. ISBN 978-1-118-99940-0. doi: 10.1002/j.2334-5837.2015.00089.x. URL http://eu.wiley.com/WileyCDA/WileyTitle/productCd-1118999401.html.

[34] A. Deursen, P. Klint, and J. Visser. Domain-specific languages: An annotated bibliography. *SIGPLAN Not.*, 35(6):26–36, June 2000. ISSN 0362-1340. doi: 10.1145/352029.352035. URL http://doi.acm.org/10.1145/352029.352035.

[35] C. Dickerson, S. Ji, and R. Roslan. Formal methods for a system of systems analysis framework applied to traffic management. In *11th System of Systems Engineering Conference (SoSE)*, pages 1–6, June 2016. doi: 10.1109/SYSOSE.2016.7542918.

[36] D.Muñante, L. Gallon, and P. Aniorté. An approach based on model-driven engineering to define security policies using orbac. In *International Conference on Availability, Reliability and Security*, pages 324–332, Sept 2013. doi: 10.1109/ARES.2013.44.

[37] H. Dogan, C. Ncube, S. L. Lim, M. Henshaw, C. Siemieniuch, M. Sinclair, V. Barot, S. Henson, M. Jamshidi, and D. DeLaurentis. Economic and societal significance of the Systems of Systems research agenda. In *IEEE International Conference on Systems, Man, and Cybernetics*, pages 1715–1720, Oct 2013. doi: 10.1109/SMC.2013.295.

[38] M. Duren, H. Aldridge, R. Abercrombie, and F. Sheldon. Designing and operating through compromise: Architectural analysis of ckms for the advanced metering infrastructure. In *Proceedings of the Eighth Annual Cyber Security and Information Intelligence Research Workshop*, CSIIRW '13, pages 48:1–48:3, 2013. ISBN 978-1-4503-1687-3. doi: 10.1145/2459976.2460031. URL http://doi.acm.org/10.1145/2459976.2460031.

[39] G. Elahi and E. Yu. A goal oriented approach for modeling and analyzing security trade-offs. In *Proceedings of the 26th International Conference on Conceptual Modeling*, ER'07, pages 375–390, 2007. ISBN 3-540-75562-4, 978-3-540-75562-3. URL http://dl.acm.org/citation.cfm?id=1784489.1784524.

[40] I. Eusgeld, C. Nan, and S. Dietz. System-of-system approach for interdependent critical infrastructures. *Reliability Engineering & System Safety*, 96(6):679 – 686, 2011. ISSN 0951-8320. doi: http://dx.doi.org/10.1016/j.ress.2010.12.010. URL http://www.sciencedirect.com/science/article/pii/S0951832010002668. ESREL Special Issue.

[41] O. Faldik, R. Payne, J. Fitzgerald, and B. Buhnova. Modelling system of systems interface contract behaviour. In *11th International Workshop on Formal Engineering approaches to Software Components and Architectures (FESCA)*, pages 1–15, 2017. doi: 10.4204/EPTCS.245.1.

[42] K. Falkner, C. Szabo, V. Chiprianov, G. Puddy, M. Rieckmann, D. Fraser, and C. Aston. Model-driven performance prediction of systems of systems. *Software & Systems Modeling, soSym*, Jul 2016. ISSN 1619-1374. doi: 10.1007/s10270-016-0547-8. URL https://doi.org/10.1007/s10270-016-0547-8.

[43] R. France and B. Rumpe. Model-driven development of complex software: A research roadmap. In *Future of Software Engineering*, FOSE '07, pages 37–54, 2007. ISBN 0-7695-2829-5. doi: 10.1109/FOSE.2007.14. URL http://dx.doi.org/10.1109/FOSE.2007.14.

[44] S. Friedenthal, A. Moore, and R. Steiner. *A Practical Guide to SysML, Third Edition: The Systems Modeling Language*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition, 2014. ISBN 0128002026, 9780128002025.

[45] L. Garcas, A. Ampatzoglou, P. Avgeriou, and E. Nakagawa. Quality attributes and quality models for ambient assisted living software systems: A systematic mapping. *Information and Software Technology*, 82:121 – 138, 2017. ISSN 0950-5849. doi: http://dx.doi.org/10.1016/j.infsof.2016.10.005. URL http://www.sciencedirect.com/science/article/pii/S0950584916302932.

[46] V. Gehlot, S. Kulkarni, and J. Brzozowski. Modeling and performance simulation of a software architecture for large-scale measurement of broadband networks using colored petri nets. In *Proceedings of the 19th Communications & Networking Symposium*, CNS '16, pages 4:1–4:8, 2016. ISBN 978-1-5108-2317-4. URL http://dl.acm.org/citation.cfm?id=2962686.2962690.

[47] 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*, 28:720 – 727, 2014. ISSN 1877-0509. doi: http://dx.doi.org/10.1016/j.procs.2014.03.086. URL http://www.sciencedirect.com/science/article/pii/S1877050914001495. Conference on Systems Engineering Research.

[48] C. Guariniello and D. DeLaurentis. Supporting design via the system operational dependency analysis methodology. *Research in Engineering Design*, 28(1):53–69, 2016. ISSN 1435-6066. doi: 10.1007/s00163-016-0229-0. URL https://doi.org/10.1007/s00163-016-0229-0.

[49] M. Guessi, E. Cavalcante, and L. Oliveira. Characterizing architecture description languages for software-intensive systems-of-systems. In *Proceedings of the Third International Workshop on Software Engineering for Systems-of-Systems*, SESoS '15, pages 12–18, 2015. URL http://dl.acm.org/citation.cfm?id=2821418.2821422.

[50] M. Guessi, V. Neto, T. Bianchi, K. Felizardo, F. Oquendo, and E.Nakagawa. A systematic literature review on the description of software architectures for systems of systems. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, SAC '15, pages 1433–1440, 2015. ISBN 978-1-4503-3196-8. doi: 10.1145/2695664.2695795. URL http://doi.acm.org/10.1145/2695664.2695795.

[51] V. Gupta and G. Lehal. A survey of text mining techniques and applications. *Journal of Emerging Technologies in Web Intelligence*, 1(1):60–76, 2009. URL www.scopus.com.

[52] J. El Hachem. Towards model driven architecture and analysis of system of systems access control. In *IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 2, pages 867–870, May 2015. doi: 10.1109/ICSE.2015.280.

[53] J. El Hachem, V. Chiprianov, A. Babar, and P. Aniorte. Towards methodological support for secure architectures of software-intensive systems-of-systems. In *Post-proceedings of the International Colloquium on Software-intensive Systems-of-Systems (ICSoS)-ECSA'16*, Copenhagen, Denmark, 2016.

[54] J. EL Hachem, Z. Pang, V. Chiprianov, A. Babar, and P. Aniorte. Model driven software security architecture of systems-of-systems. In *23rd Asia-Pacific Software Engineering Conference (APSEC)*, pages 89–96, Dec 2016. doi: 10.1109/APSEC.2016.023.

[55] J. EL Hachem, T. Al 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*, 2017.

[56] W. Harrison. The role of graph theory in system of systems engineering. *IEEE Access*, 4:1716–1742, 2016. ISSN 2169-3536. doi: 10.1109/ACCESS.2016.2559450.

[57] 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*, 60:1604 – 1612, 2015. ISSN 1877-0509. doi: http://dx.doi.org/10.1016/j.procs.2015.08.270. URL http://www.sciencedirect.com/science/article/pii/S1877050915023972.

[58] Intel industry. Security practices for smart buildings: Good, better, best. 2016. URL https://www.intel.com/content/dam/www/public/us/en/documents/solution-briefs/security-practices-smart-buildings-brief.pdf.

[59] R. Irfan, C. King, D. Grages, S. Ewen, S. Khan, S. Madani, J. Kolodziej, L. Wang, D. Chen, A. Rayes, N. Tziritas, C. Xu, A. Zomaya, A. Alzahrani, and H. Li. A survey on text mining in social networks. *Knowledge Engineering Review*, 30(2):157–170, 2015. URL www.scopus.com.

[60] International Organization for Standardization ISO. Standards for systems and software engineering - architectural description. 2011. URL http://cabibbo.dia.uniroma3.it/asw/altrui/iso-iec-ieee-42010-2011.pdf.

[61] International Organization for Standardization ISO. Standards for information technology - security techniques - information security management systems. 2014. URL https://www.iso.org/standard/63411.html.

[62] International Organization for Standardization ISO. Standards for systems and software engineering - system life cycle processes. 2015. URL https://www.iso.org/standard/63711.html.

[63] M. Jamshidi. *Systems of Systems engineering principles and applications*. John Wiley and Sons, 2009. ISBN 9780470403501. doi: 10.1002/9780470403501. URL http://onlinelibrary.wiley.com/book/10.1002/9780470403501.

[64] J. Jürjens. *UMLsec: Extending UML for Secure Systems Development*, pages 412–425. Springer Berlin Heidelberg, 2002. ISBN 978-3-540-45800-5. doi: 10.1007/3-540-45800-X_32. URL https://doi.org/10.1007/3-540-45800-X_32.

[65] k. Yskout, R. Scandariato, and W. Joosen. Do security patterns really help designers? In *Proceedings of the 37th International Conference on Software Engineering - Volume 1*, ICSE, pages 292–302, 2015. ISBN 978-1-4799-1934-5. URL http://dl.acm.org/citation.cfm?id=2818754.2818792.

[66] D. Kimura, T. Osaki, K. Yanoo, S. Izukura, H. Sakaki, and A. Kobayashi. Evaluation of it systems considering characteristics as system of systems. In *6th International Conference on System of Systems Engineering*, pages 43–48, June 2011. doi: 10.1109/SYSOSE.2011.5966571.

[67] B. Kitchenham, R. Pretorius, D. Budgen, O. Brereton, M. Turner, M. Niazi, and S. Linkman. Systematic literature reviews in software engineering: A tertiary study. *Information and Software Technology*, 52(8):792 – 805, 2010. ISSN 0950-5849. doi: https://doi.org/10.1016/j.infsof.2010.03.006. URL http://www.sciencedirect.com/science/article/pii/S0950584910000467.

[68] J. Klein and van H. Vliet. A systematic review of system-of-systems architecture research. In *Proceedings of the 9th International ACM Sigsoft Conference on Quality of Software Architectures*, QoSA '13, pages 13–22, 2013. ISBN 978-1-4503-2126-6. doi: 10.1145/2465478.2465490. URL http://doi.acm.org/10.1145/2465478.2465490.

[69] A. Kobetski and J. Axelsson. Towards safe and secure systems of systems: Challenges and opportunities. In *Proceedings of the Symposium on Applied Computing*, SAC, pages 1803–1806, 2017. ISBN 978-1-4503-4486-9. doi: 10.1145/3019612.3028252. URL http://doi.acm.org/10.1145/3019612.3028252.

[70] 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)*, pages 214–219, May 2015. doi: 10.1109/SYSOSE.2015.7151925.

[71] K. Kravari and N. Bassiliades. A survey of agent platforms. *Journal of Artificial Societies and Social Simulation*, 18(1):11, 2015. ISSN 1460-7425. doi: 10.18564/jasss.2661. URL http://jasss.soc.surrey.ac.uk/18/1/11.html.

[72] I. Kurtev, J. Bézivin, F. Jouault, and P. Valduriez. Model-based DSL frameworks. In *Companion to the 21st ACM SIGPLAN Symposium on Object-oriented Programming Systems, Languages, and Applications*, OOPSLA '06, pages 602–616, 2006. ISBN 1-59593-491-X. doi: 10.1145/1176617.1176632. URL http://doi.acm.org/10.1145/1176617.1176632.

[73] T. Lodderstedt, D. Basin, and J. Doser. Secureuml: A uml-based modeling language for model-driven security. In *Proceedings of the 5th International Conference on The Unified Modeling Language*, UML '02, pages 426–441, 2002. ISBN 3-540-44254-5. URL http://dl.acm.org/citation.cfm?id=647246.719477.

[74] L. Lucio, Q. Zhang, H. Nguyen, M. Amrani, J. Klein, H. Vangheluwe, and Y. Traon. *Advances in Model-Driven Security*. Advances in Computers series, Volume 93, 2014. ISBN 9780128001622. URL http://hdl.handle.net/10993/10204.

[75] M. Maier. Architecting principles for systems-of-systems. *Systems Engineering*, 1 (4):267–284, 1998. ISSN 1520-6858. doi: 10.1002/(SICI)1520-6858(1998)1:4<267::AID-SYS3>3.0.CO;2-D. URL http://dx.doi.org/10.1002/(SICI)1520-6858(1998)1:4<267::AID-SYS3>3.0.CO;2-D.

[76] N. Medvidovic and R. Taylor. A classification and comparison framework for software architecture description languages. *IEEE Transactions on Software Engineering*, 26 (1):70–93, Jan 2000. ISSN 0098-5589. doi: 10.1109/32.825767.

[77] 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*, 5(2):1–19, 2014. ISSN 1947-3036. doi: 10.4018/ijsse.2014040101. URL http://dx.doi.org/10.4018/ijsse.2014040101.

[78] M. Merabti, M. Kennedy, and W. Hurst. Critical infrastructure protection: A 21st century challenge. In *International Conference on Communications and Information Technology (ICCIT)*, pages 1–6, March 2011. doi: 10.1109/ICCITECHNOL.2011.5762681.

[79] M. Merabti, B.Alohali, and K. Kifayat. A new key management scheme based on smart grid requirements. In *In Proceedings of the 9th International Conference on Computer Engineering and Applications (CEA)*, page 436–443, March 2015. doi: 10.1109/ICCITECHNOL.2011.5762681.

[80] E. Merks and D. Steinberg. Models to code with eclipse modeling framework. *Eclipse Conference*, tutorials session, 2005. ISSN 1520-6858. URL http://www.eclipsecon.org/2005/presentations/EclipseCon2005_Tutorial11final.pdf.

[81] M. Mohsin, Z. Anwar, G. Husari, E. Al-Shaer, and M. A. Rahman. IoTSAT: A formal framework for security analysis of the internet of things (IoT). In *IEEE Conference on Communications and Network Security (CNS)*, pages 180–188, Oct 2016. doi: 10.1109/CNS.2016.7860484.

[82] P. Molina. Introduction to model driven software development, code generation. *SlideShare presentation, https://goo.gl/images/4p5MpV, 2010*.

[83] 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)*, pages 1–6, June 2016. doi: 10.1109/SYSOSE.2016.7542895.

[84] M. Mori, A. Ceccarelli, P. Lollini, B. Frömel, F. Brancati, and A. Bondavalli. Systems-of-systems modeling using a comprehensive viewpoint-based SysML profile. *Journal of Software: Evolution and Process*, 2017. ISSN 2047-7481. doi: 10.1002/smr.1878. URL http://dx.doi.org/10.1002/smr.1878.

[85] H. Mouratidis, M. Kolp, P. Giorgini, and S. Faulkner. An architectural description language for secure multi-agent systems. *Web Intelligent and Agent System*, 8(1): 99–122, January 2010. ISSN 1570-1263. doi: 10.3233/WIA-2010-0182. URL http://dx.doi.org/10.3233/WIA-2010-0182.

[86] G. Muller and C. Dagli. Simulation for a coevolved system-of-systems meta-architecture. In *11th System of Systems Engineering Conference (SoSE)*, pages 1–6, June 2016.

[87] E. Nakagawa, M. Gonçalves, M. Guessi, L. Oliveira, and F. Oquendo. The state of the art and future perspectives in systems of systems software architectures. In *Proceedings of the First International Workshop on Software Engineering for Systems-of-Systems*, SESoS '13, pages 13–20, 2013. ISBN 978-1-4503-2048-1. doi: 10.1145/2489850.2489853. URL http://doi.acm.org/10.1145/2489850.2489853.

[88] V. Neto. Validating Emergent Behaviors in Systems-of-Systems through Model Transformations. In *Proceedings of the ACM PhD Student Research Competition at MODELS co-located with the 19th International Conference on Model Driven Engineering Languages and Systems*, St. Malo, France, October 2016. URL https://hal.archives-ouvertes.fr/hal-01443187.

[89] V. Neto and E. Nakagawa. A biological inspiration to support emergent behavior in systems-of-systems development. In *9th Workshop on Distributed Software Development, Software Ecosystems and Systems-of-Systems - WDES"*, pages 33–40, September 2015. URL http://www.lbd.dcc.ufmg.br/bdbcomp/servlet/Trabalho?id=23563.

[90] P. Nguyen, S. Ali, and T. Yue. Model-based security engineering for cyber-physical systems: A systematic mapping study. *Information and Software Technology*, 83:116 – 135, 2017. ISSN 0950-5849. doi: http://dx.doi.org/10.1016/j.infsof.2016.11.004. URL http://www.sciencedirect.com/science/article/pii/S0950584916303214.

[91] 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)*, pages 1–6, June 2016. doi: 10.1109/SYSOSE.2016.7542908.

[92] 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*, 48(2), 2015. ISSN 15577341, 03600300. doi: 10.1145/2794381.

[93] N.Jennings, K. Sycara, and M. Wooldridge. A roadmap of agent research and development. *Autonomous Agents and Multi-Agent Systems*, 1(1):7–38, 1998. ISSN 1387-2532. doi: 10.1023/A:1010090405266. URL https://doi.org/10.1023/A:1010090405266.

[94] P. Oberndorf and C. Sledge. Evolution of a software engineer in a SoS system engineering world. In *IEEE International Systems Conference*, pages 91–96, April 2010. doi: 10.1109/SYSTEMS.2010.5482478.

[95] Office of the Deputy Under Secretary of Defense for Acquisition, Systems Technology, and Software Engineering. Systems engineering guide for systems of systems. Technical report, Version 1.0. Washington, DC: ODUSD(A&T)SSE, 2008. URL http://www.acq.osd.mil/se/docs/SE-Guide-for-SoS.pdf.

[96] O. Ogunnusi and S. Razak. Attacks and security solutions for agent communication in multi-agent systems. *Proceedings of International Journal of Soft Computing*, 10: 99–109, 2015. doi: 10.3923/ijscomp.2015.99.109. URL http://medwelljournals.com/abstract/?doi=ijscomp.2015.99.109.

[97] Object Management Group OMG. System modeling language specifications, version 1.4. 2015. URL http://www.omg.org/spec/SysML/1.4/.

[98] Object Management Group OMG. Unified modeling language specifications, version 2.5. 2015. URL http://www.omg.org/spec/UML/2.5/.

[99] F. Oquendo. *Software Architecture Challenges and Emerging Research in Software-Intensive Systems-of-Systems*, pages 3–21. Springer International Publishing, 2016. ISBN 978-3-319-48992-6. doi: 10.1007/978-3-319-48992-6_1. URL http://dx.doi.org/10.1007/978-3-319-48992-6_1.

[100] F. Oquendo. Formally describing the software architecture of systems-of-systems with sosadl. In *11th System of Systems Engineering Conference (SoSE), Kongsberg, Norway*, pages 1–6, June 2016. doi: 10.1109/SYSOSE.2016.7542926.

[101] F. Oquendo, J. Leite, and T. Batista. Specifying architecture behavior with sysadl. In *13th Working IEEE/IFIP Conference on Software Architecture (WICSA)*, pages 140–145, April 2016. doi: 10.1109/WICSA.2016.40.

[102] F. Oquendo, J. Leite, and T. Batista. Executing software architecture descriptions with sysadl. In *Proceedings of the 10th European Conference on Software Architecture, ECSA, Copenhagen, Denmark*, pages 129–137. Springer International Publishing, 2016. ISBN 978-3-319-48992-6. doi: 10.1007/978-3-319-48992-6_9. URL https://doi.org/10.1007/978-3-319-48992-6_9.

[103] J. Pavón, J. Gómez-Sanz, and A. López Paredes. The sicossys approach to sos engineering. In *6th International Conference on System of Systems Engineering*, pages 179–184, June 2011. doi: 10.1109/SYSOSE.2011.5966594.

[104] M. Peacock. An analysis of security issues in building automation systems. In *12th Australian Information Security Management Conference, Perth, Western Australia*, December 2014. doi: 10.4225/75/57b691dfd9386. URL http://ro.ecu.edu.au/ism/170.

[105] J. Pérez, J. Díaz, J. Garbajosa, A. Yagüe, E. Gonzalez, and M. Lopez-Perea. Large-scale smart grids as system of systems. In *Proceedings of the First International Workshop on Software Engineering for Systems-of-Systems*, SESoS, pages 38–42, 2013. ISBN 978-1-4503-2048-1. doi: 10.1145/2489850.2489858. URL http://doi.acm.org/10.1145/2489850.2489858.

[106] M. Plachkinova, A. Vo, and A. Alluhaidan. Emerging trends in smart home security, privacy, and digital forensics. In *Proceedings of the AMERRICAS conference on information systems (AMCIS)*, 2016. ISBN 978-0-9966831-2-8. URL http://aisel.aisnet.org/amcis2016/ITProj/Presentations/23/.

[107] S. Posland and M. Calisti. Towards improved trust and security in fipa agent platforms. *in Autonomous Agents*, 2000. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.464.8554&rep=rep1&type=pdf.

[108] O. Ramón, F. Molina, J. Molina, and J. Álvarez. ModelSec: A generative architecture for model-driven security. *Journal of Universal Computer Science, UCS*, 15:2957–2980, 2009. doi: 10.3217/jucs-015-15-2957. URL http://www.jucs.org/jucs_15_15/modelsec_a_generative_architecture.

[109] R. Ramsin and R. Paige. Process-centered review of object oriented software development methodologies. *ACM Comput. Surv.*, 40(1), February 2008. ISSN 0360-0300. doi: 10.1145/1322432.1322435. URL http://doi.acm.org/10.1145/1322432.1322435.

[110] A. Rashid, S. Naqvi, R. Ramdhany, M. Edwards, R. Chitchyan, and M. Babar. Discovering unknown known security requirements. In *IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, pages 866–876, May 2016. doi: 10.1145/2884781.2884785.

[111] S. Rosen, J. Ramsey, C. Harvey, and S. Guharay. Efficient analysis for emergency management using simulation metamodeling: A case study for a medical trauma center. volume 47, pages 9–16, 2015. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-84954128520&partnerID=40&md5=797727f98f579f5f44afc0de6e94646f.

[112] I. Rudowsky. Intelligent agents. *Journal of Communications of the Association for Information Systems (CAIS)*, 14(14), 2004. ISSN 1529-3181. URL http://aisel.aisnet.org/cais/vol14/iss1/14.

[113] D. Rumsfeld. *Known and Unknown: A Memoir*, page 832. Penguin Group USA, 2011. ISBN 978-1-59523-067-6. URL http://www.penguinrandomhouse.com/books/304431/known-and-unknown-by-donald-rumsfeld/9781595230843/.

[114] FIPA security SIG request for information. Fipa. 2001. URL http://www.fipa.org.

[115] B. Selic. The theory and practice of modeling language design for model-based software engineering: A personal perspective. In *Proceedings of the 3rd International Summer School Conference on Generative and Transformational Techniques in Software Engineering III*, GTTSE'09, pages 290–321, 2011. ISBN 3-642-18022-1, 978-3-642-18022-4. URL http://dl.acm.org/citation.cfm?id=1949925.1949933.

[116] R. Sparrow, A. Adekunle, R. Berry, and R. Farnish. Study of two security constructs on throughput for wireless sensor multi-hop networks. In *38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 1302–1307, May 2015. doi: 10.1109/MIPRO.2015.7160476.

[117] C. Szabo and L. Birdsey. *Validating Emergent Behavior in Complex Systems*, pages 47–62. Springer International Publishing, Cham, 2017. ISBN 978-3-319-64182-9. doi: 10.1007/978-3-319-64182-9_4. URL https://doi.org/10.1007/978-3-319-64182-9_4.

[118] R. Taylor, N. Medvidovic, and E. Dashofy. *Software Architecture: Foundations, Theory, and Practice*. John Wiley and Sons, 2009. ISBN 9780470167748. URL http://eu.wiley.com/WileyCDA/WileyTitle/productCd-EHEP000180.html.

[119] independent security news website The Security Ledger. Ibm research calls out smart building risks. 2016. URL https://securityledger.com/2016/02/ibm-research-calls-out-smart-building-risks/.

[120] J. Tong, W. Sun, and L. Wang. *A Smart Home Network Simulation Testbed for Cybersecurity Experimentation*, pages 136–145. 2014. ISBN 978-3-319-13326-3. doi: 10.1007/978-3-319-13326-3_14. URL https://doi.org/10.1007/978-3-319-13326-3_14.

[121] United-Nation Department of economics UN and social affairs. World urbanization prospects. 2014.

[122] A. Vallecillo. On the combination of domain specific modeling languages. In *Proceedings of the 6th European Conference on Modelling Foundations and Applications*, ECMFA, pages 305–320, 2010. ISBN 3-642-13594-3, 978-3-642-13594-1. doi: 10.1007/978-3-642-13595-8_24. URL http://dx.doi.org/10.1007/978-3-642-13595-8_24.

[123] X. Vila, A. Schuster, and A. Riera. Security for a multi-agent system based on JADE. *Computer Security*, 26(5):391–400, August 2007. ISSN 0167-4048. doi: 10.1016/j.cose.2006.12.003. URL http://dx.doi.org/10.1016/j.cose.2006.12.003.

[124] S. Vitabile, V. Conti, C. Militello, and F. Sorbello. An extended JADE-S based framework for developing secure multi-agent systems. *Computer Standards & Interfaces*, 31(5):913 – 930, 2009. ISSN 0920-5489. doi: http://dx.doi.org/10.1016/j.csi.2008.03.017. URL http://www.sciencedirect.com/science/article/pii/S0920548908000378. Specification, Standards and Information Management for Distributed Systems.

[125] D. Wachholder and C. Stary. Enabling emergent behavior in systems-of-systems through bigraph-based modeling. In *10th System of Systems Engineering Conference (SoSE)*, pages 334–339, May 2015. doi: 10.1109/SYSOSE.2015.7151954.

[126] M. Wooldridge. *An Introduction to MultiAgent Systems*. John Wiley & Sons, 2nd edition, 2009. ISBN 0470519460, 978-04705194625.

[127] R.K. Yin. Case study research: Design and methods. In *Sage publications, third edition*, Italy, 2003.

[128] G. You, X. Sun, M. Sun, J. Wang, and Y. Chen. Bibliometric and social network analysis of the SoS field. In *9th International Conference on System of Systems Engineering (SOSE)*, pages 13–18, June 2014. doi: 10.1109/SYSOSE.2014.6892456.

[129] B. Zeigler and H. Sarjoughian. *Guide to Modeling and Simulation of Systems of Systems*. Springer-Verlag London, 2012. ISBN 978-1-4471-6933-8. doi: 10.1007/978-0-85729-865-2. URL http://www.springer.com/la/book/9780857298645.

[130] L. Zhang. *Applying System of Systems Engineering Approach to Build Complex Cyber Physical Systems*, pages 621–628. Springer International Publishing, 2015. ISBN 978-3-319-08422-0. doi: 10.1007/978-3-319-08422-0_88. URL https://doi.org/10.1007/978-3-319-08422-0_88.

[131] W. Zhang, Z. Li, W. Wang, and Q. Li. System of systems safety analysis of gnss based on functional dependency network analysis. *International Journal of Applied Mathematics and Information Sciences*, 10(6):2227–2235, 2016. doi: doi:10.18576/amis/100625. URL http://dx.doi.org/10.18576/amis/100625.