The Journal of Systems and Software 162 (2020) 110484

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Modeling, analyzing and predicting security cascading attacks in smart buildings systems-of-systems

Jamal EL Hachem[a,*], Vanea Chiprianov[a], Muhammad Ali Babar[b], Tarek AL Khalil[c], Philippe Aniorte[a]

[a] UNIV PAU & PAYS ADOUR/E2S UPPA, LIUPPA, EA3000, MONT DE MARSAN, 40000, FRANCE
[b] UNIVERSITY OF ADELAIDE, Adelaide, Australia
[c] ANTONINE UNIVERSITY, Baabda, Lebanon

ARTICLE INFO

Article history:
Received 31 May 2018
Revised 29 September 2019
Accepted 25 November 2019
Available online 3 December 2019

MSC:
00-01
99-00

Keywords:
Systems-of-systems
Security modeling and analysis
Model driven engineering
Software architecture
Multi-agent systems simulation
Smart buildings

ABSTRACT

Software systems intelligence and complexity have been continuously increasing to deliver more and more features to support business critical and mission critical processes in numerous domains such as defense, health-care, and smart cities. Contemporary software-based solutions are composed of several software systems, that form System-of-Systems (SoS). SoS differentiating characteristics, such as emergent behavior, introduce specific issues that render their security modeling, simulation and analysis a critical challenge. The aim of this work is to investigate how Software Engineering (SE) approaches can be leveraged to model and analyze secure SoS solutions for predicting high impact (cascading) attacks at the architecture stage. In order to achieve this objective, we propose a Model Driven Engineering method, Systems-of-Systems Security (SoSSec), that comprises: (1) a modeling language (SoSSecML) for secure SoS modeling and (2) Multi-Agent Systems (MAS) for security analysis of SoS architectures. To illustrate our proposed approach in terms of modeling, simulating, and discovering attacks, we have conducted a case study on a real-life smart building SoS, the Adelaide University Health and Medical School (AHMS). The results from this case study demonstrate that our proposed method discovers cascading attacks comprising of a number of individual attacks, such as a Denial of Service, that arise from a succession of exploited vulnerabilities through interactions among the constituent systems of SoS. In future work, we intend to extend SoSSec to address diverse unknown emergent behaviors and non-functional properties such as safety and trust.

1. Introduction

The SoS concept has been used since 1950's to describe complex Systems that are created by assembling various distributed, independent Systems that interact to accomplish a common goal (Nielsen et al., 2015). SoS 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, the integration of widespread CSs that interact to achieve the SoS goal(s) leading to potentially expected or unexpected beneficial emergent behaviors, as well as detrimental emergent behaviors (Kopetz et al., 2015). Moreover, even if the properties of each CS are known and well defined, engineering the whole SoS and predicting its functional and non-functional properties remains a challenging task.

SoS engineering has drawn increasing attention since SoS application domains and security concerns became associated with many of our contemporary society systems such as defense, emergency management and response, health-care, smart cities and E-commerce. Therefore, there is a growing interest in SoS, their architecture and specially their security. The SoS domain continues receiving more and more attention from the research community as shown by the increasing number of international conferences and workshops (You et al., 2014). Despite all these efforts, numerous projects studying the SoS development agenda for the com-

* Corresponding author.
E-mail addresses: jamal.elhachem@univ-pau.fr (J.E. Hachem), vanea.chiprianov@univ-pau.fr (V. Chiprianov), ali.babar@adelaide.edu.au (M.A. Babar), philippe.aniorte@univ-pau.fr (P. Aniorte).

https://doi.org/10.1016/j.jss.2019.110484
0164-1212/© 2019 Elsevier Inc. All rights reserved.

ing years, such as CPSoS[1] (2013–2016), Road2SoS[2] (2011–2013), T-AREA-SoS[3] (2011–2013) and several systematic literature reviews (Dahmann and Roedler, 2016; Nielsen et al., 2015; Guessi et al., 2015; Klein and van H., 2013) emphasize that the research in the SoS field is still at early stages and there is still a need of bridging different software engineering solutions and proposing accurate methods, languages and tools to address SoS complexity and non-functional properties (in particular security) specially at the software architecture level.

Indeed, SoS possess specific characteristics, in particular: the emergent behavior, the operational and managerial independence of the CSs and the geographic distribution of these CSs (Maier, 1998). These specific characteristics introduce many Software Engineering (SE) challenges not only related to their functional properties, but to their non-functional properties as well, specially security, on which we are focusing in this paper. In addition to traditional security problems targeting their CSs, SoS suffer from challenges arising from their specific characteristics, as reported for example in Hachem (2015) and Chiprianov et al. (2014).

It is important to address these challenges early at the architecture phase to avoid time and cost wastage of later changes and to prevent massive damages targeting the SoS functionalities and impacting people's safety and security. To address SoS modeling challenges, one of the most effective SE approaches is Model Driven Engineering (MDE) (Nakagawa et al., 2013). The fundamental advantage of MDE is that it allows to define the system models separately from the implementation/platform while ensuring a smooth transition between the models' specification and their execution through the (semi)automatic generation of code and tools (Kurtev et al., 2006).

On the other hand, simulation plays a fundamental role in analyzing SoS dynamic/behavioral aspects, specially to investigate the CS interactions and security emergent behaviors and their impact on the SoS (Baldwin et al., 2015). Multi-Agent Systems (MAS) is a simulation approach that is widely used to analyze complex, distributed and critical systems (Bellifemine et al., 2007). In addition, as we will develop later in Section 4.3.1, MAS concepts are similar to SoS concepts, thus they can be used to express the execution semantics of the SoS structural (CSs and their interfaces) and behavioral aspects (CS interactions and security emergent behaviors).

The reported research has been motivated by the following Research Questions (RQs):

- *RQ1: How to model SoS structure and behavior with a particular focus on security aspects?*
- *RQ2: How to analyze these models to discover emergent security issues, precisely the cascading attacks?*

We argue that MDE and MAS approaches can be leveraged to model and analyze secure SoS architectures. In order to answer the RQ1, we propose an architecture description language (SoSSecML) and modeling tool (graphical editor implemented as a Papyrus extension) to design secure SoS architectures; and to investigate our RQ2, we offer an extension of a MAS simulation platform to analyze security issues arising from SoS constituent systems interactions. The mapping between the modeling and simulation is ensured by a generator tool: a set of transformation rules that perform the automatic generation of code from the designed models. One of the main advantages of SoSSec is its iterative feature, based on the mapping between the modeling and simulation phases, thereby allowing the analysis of several secure SoS architecture alternatives until reaching an acceptable security level. To investigate the effectiveness of our method, we employed a small-scale view of a Smart City SoS: a smart building. Following a structured protocol, we applied our method on a real-world study domain: the ***A****delaide University* ***H****ealth and* ***M****edical* ***S****chool* (AHMS) smart building. Results show: 1) the relative ease of use of the proposed method (SoSSecML and modeling and analysis tools), 2) the ability of the proposed language in modeling the AHMS smart building as an SoS composed of several CSs, taking into account SoS specific characteristics and the cascading attack security elements, 3) SoSSec ability in discovering and analyzing one of the specific SoS security issues: the unknown cascading attacks, as testified by the AHMS research team.

The structure of this article is organized as follows: In Section 2 we introduce the SoS specific security issue: the cascading attack. Section 3 investigates the related work for secure SoS modeling and analysis. Section 4 briefly presents our proposed method (SoSSec). Sections 5 and 6 show the results of its application on the AHMS smart building study domain. Section 7 discusses our results and Section 8 concludes the paper.

2. SoS cascading attack security issue

SoS solutions are becoming more open and prone to attacks because of their interactive and collaborative nature. The complexity of their architectures, emergent behavior and interconnections between their independent CSs can make them vulnerable to a variety of traditional and emergent security attack. In our work, we focus on a specific security issue, the *security cascading attack*, which has the potential of huge impact in the context of SoS. This kind of security issue can affect the whole SoS resulting in damages ranging from the interruption of SoS functionalities and services, to human-related security problems. The Dyn Cyberattack in 2016,[4] the Stuxnet in 2010,[5] the Northeast Blackout in 2003[6] and the Google building attack in 2013[7] are some examples of well known cascading attacks that caused catastrophic impacts effects on the targeted SoS.

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 CSs together (SoS interactions) intensifies the attack and magnifies its effects. In fact, many of the CSs may suffer from small, known and unresolved (because of financial or other considerations) 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. 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 (Guariniello and DeLaurentis, 2014).

Therefore, we define a *cascading attack* as: "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 differ-

[1] Towards a European roadmap on research and innovation in engineering and management of Cyber-Physical SoS.

[2] Development of strategic research and engineering roadmaps in SoS Engineering and related case studies.

[3] Trans-Atlantic Research and Education Agenda in SoS.

[4] http://money.cnn.com/2016/10/22/technology/cyberattack-dyn-ddos/index.html, accessed on March 2018.

[5] http://large.stanford.edu/courses/2015/ph241/holloway1/, accessed on March 2018.

[6] https://www.scientificamerican.com/article/2003-blackout-five-years-later/, accessed on March 2018.

[7] https://www.wired.com/2013/05/googles-control-system-hacked/, accessed on March 2018.

ent CSs, and connected through the interactions between these CSs when collaborating to accomplish the SoS global goal(s)".

Model-based security engineering or security by design is a key solution to encounter this challenge (Nguyen et al., 2017). Security should be integrated and analyzed during the design of SoS and not in the aftermath to avoid extensive functional, social and economical damages. In this article, we focus on secure SoS modeling, simulation and analysis to discover potential cascading attacks emergent behaviors, early at the architecture level of the SoS development life cycle.

3. An overview of research on SoS security-sensitive architecture modeling and analysis

SoS engineering relies significantly on modeling and simulation, as shown by recent studies (Baldwin et al., 2015; Nielsen et al., 2015). Nguyen et al. review, in a recent systematic mapping study (Nguyen et al., 2017), model based approaches for security engineering in Cyber-Physical Systems (CPS). The results highlight a noteworthy increase in the number of studies targeting model based software engineering for CPS, as well as the effectiveness of modeling languages and automatic model transformations in this context. To conclude their study, the authors point out to several shortcomings to engineer security for CPS, mainly: The lack of solutions, tool support, application studies; as well as the challenge of bridging DSLs and integrating them with security analysis through model transformations for example.

Mori et al. (2017) propose a MDE extension of the System Modeling Language (SysML) 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 to detect security incidents and vulnerabilities. The authors mention the possibility of customizing their tool to generate code for an ulterior analysis using the hazard analysis technique. This work is still at a very early stage, only the profile and a corresponding editor have been implemented.

Mohsin et al. (2016) propose a formal framework to analyze the security of Internet of Things (IoT). They formally specify a general behavioral model and an interconnected threat propagation tree for IoT. They claim that their work is the first attempt to model and analyze IoT general behavior and threat resiliency. However, in this study the IoT is modeled in a generic way with a graph of connected nodes, without considering the independence and emergent behavior characteristics.

Due to the very limited number of primary studies that jointly cover the *modeling* and *analysis* of *SoS* considering their *security*, we explore in what follows the *secure SoS modeling* and the *secure SoS analysis* related work.

3.1. Secure SoS architecture modeling

To answer our RQ1, we gain insight into the current status of secure SoS modeling research by reviewing the existing approaches that jointly or partially address (*SoS* and/or *security*) and *modeling*.

3.1.1. SoS architecture and security modeling

Merabti et al. (2011), consider the importance of security design when architecting critical infrastructures. The study asserts the necessity of considering vulnerabilities and security issues when designing SoS. However, it does not detail a concrete way to handle it. In a more recent work (Merabti et al., 2015), they propose a cryptographic key management schema that considers smart grid security requirements. The solution is restrained to secure the data exchanges between two CSs 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 the state of the art of *SoS modeling* and systems *security modeling*.

3.1.2. SoS architecture modeling

Traditionally, many approaches were used to model 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 (ADL) 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.

Some formal methods (e.g. graph based methods: Bayesian Networks, Functional Dependency Network Analysis) seem well suited to model the geographic distribution and CS interactions but they are still limited compared to ADL regarding the representation of the structural architecture.

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 CS interactions. However, similar to 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, ADL security extensions present more useful modeling concepts than goal oriented approaches.

For these reasons, ADL seems to be a well suited basis for the extension/definition of a modeling language for secure SoS architectures modeling (Guessi et al., 2015; Nakagawa et al., 2013).

Therefore, we restrain our review to ADL. We present and analyze below the latest ADL intended to model SoS software architecture and those issued from recent International and European SoS projects like AMADEOS[8] (2013–2016), DANSE[9] (2011–2015) and COMPASS[10] (2011–2014).

Zhang (2015) introduces an approach going from SoS architecture specification (through a combination of the Architecture Analysis and Design Language (AADL), the modelica language, and formal specifications) to its simulation. However, this approach is still immature and does not concretely cover the SoS characteristics.

Cavalcante et al. (2013) extend the Acme ADL to model cloud application architectures. The proposed ADL remains a simple first-proposal language (as stated by its authors), limited for cloud SoS.

Oquendo (2016) propose a formal ADL (SosADL) along with a model driven tool-set to describe SoS software architectures. Whilst this approach provides expressive SoS concepts, it remains powerless in expressing the unknown emergent behavior. The authors intend to use event simulation to further explore their proposition.

Another ADL that has been predominantly used and extended to model SoS architectures is SysML (Friedenthal et al., 2014). SysML is actively used in recent SoS European projects (such as DANSE and COMPASS) 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 as reported in Guessi et al. (2015) and Klein and van H. (2013). However, SysML lacks concepts to cover SoS specific characteristics such as goals, CS independence and emergent behavior as well as security concepts

[8] Architecture for Multi-criticality Agile Dependable Evolutionary Open System-of-Systems.

[9] Design for Adaptability and Evolution in Systems-of-systems Engineering.

[10] COMPrehensive modeling for Advanced Systems-of-Systems.

like vulnerability, attack, pre- and post- conditions (Hachem et al., 2016).

Moreover, due to the absence of modeling languages developed to model *SoS* security, we explored some modeling language extensions that have been developed to address *system* security modeling.

3.1.3. System security modeling

UMLSec (Jürjens, 2002) and SecureUML (Lodderstedt et al., 2002) are two UML extensions proposed to express security requirements and specify authorization constraints. Secure i* (Elahi and Yu, 2007) is a goal oriented approach intended to model and analyze security trade-offs. MoDELO (Muñante et al., 2013) is a MOdel-Driven sEcurity poLicy approach that extends UMLSec to model and evaluate access control requirements. SysML-Sec (Apvrille et al., 2016) is a SysML based model driven approach for security. It is a formal approach for requirement analysis.

The analysis of these approaches shows that none of them demonstrates modeling ability that subsequently allows the discovery of cascading attack emergent behaviors. However, some concepts from each approach should be considered by any forthcoming modeling language aiming for secure SoS architectures modeling, such as: threat, vulnerability, attack and goal. Furthermore, the vulnerability concept should be refined and additional related concepts should be introduced to explicitly model CS vulnerabilities in a way so as to allow the discovery of their sequence, to predict possible SoS cascading attacks.

3.1.4. Discussion

Results show that in recent years, diverse studies in the field of SoS have been published, many of them highlighting the urgency of considering security when engineering SoS early in the development life-cycle (Nguyen et al., 2017; Garcas et al., 2017; Duren et al., 2013). 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. Moreover, among systems and software engineering techniques that have been extended and adapted to effectively develop SoS, MDE has gained a noticeable interest with considerable evidence of activity in both SoS research and practice area (Nielsen et al., 2015). MDE is a promising approach to evaluate non-functional properties, such as security, in the early design phase (Kimura et al., 2011).

All these reasons justify the need of a new modeling language or extending an existing modeling language such as SysML to model SoS taking into account the security specific properties. The extension should introduce security concepts to allow architectural level security modeling and analysis of SoS, specially that of the cascading attack in the context of emergent behavior.

Furthermore, in order to get the maximum value from MDE approaches, there is a need of well-defined mappings between the models and their realization (Nielsen et al., 2015). ADL lack concepts to deal with models execution (Neto, 2016). Thus, there is a need of an approach that execute/simulate the secure SoS models produced with the ADL extension.

3.2. Secure SoS architecture analysis

To answer our RQ2, we conducted a review of the existing approaches that jointly or partially address (*SoS* and/or *security*) and *analysis*.

Results show that 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 (Nielsen et al., 2015; Muller and Dagli, 2016). Several initiatives have been proposed to address SoS simulation and analysis, but only a few studies consider SoS security. Therefore, we examine the strengths and limitations of each of the studied approach to addressing the SoS characteristics, in particular the emergent behavior (cascading attacks) resulting from the CS interactions. Moreover, even if the number of these studies is limited and their basis are diverse, we regrouped them, based on common features, in the following sub-categories: 1) Approaches based on threat analysis processes; 2) Graph-based approaches and 3) Goal-oriented/Agent-based approaches.

3.2.1. Approaches based on threat analysis processes

Kobetski and Axelsson investigate in Kobetski and Axelsson (2017) 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. This recent work discusses general approaches without detailing neither the methods and techniques to be used, nor the required guidance about how the possible losses and the 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 independence characteristics.

Mori et al. (2016) 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 known threats and/or vulnerabilities that can arise from this evolution. 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.

3.2.2. Graph-based approaches

Nicklas et al. (2016) 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 specific methods for each step as well as to implement dedicated tools.

In Guariniello and DeLaurentis (2014), Guariniello et al. modified the Functional Dependency Network Analysis (FDNA) to make it applicable for SoS to analyze the impact of cyber attacks on the SoS interdependencies in terms of 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 Guariniello and DeLaurentis (2014), Dahmann et al. (2013) propose a general framework to assess the security risk of a single security incident on multiple CS. The authors propose the use of the FDNA to measure the SoS missions operational effectiveness. The analysis intends to assess the ability of an SoS to operate effectively if one or more CS fail.

3.2.3. Goal-oriented/agent-based approaches

Abercrombie and Sheldon (2015) use the dynamic Agent Based Game Theoretic simulations to analyze smart grids information security by assessing the risk in terms of the probability of a successful attack. Even if this approach seems promising to calculate the probability of successful attacks at a specific time of a simulated scenario, 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 Meland et al. (2014) an approach to analyze the threats at the requirement level of the SoS development. They extend an agent-based modeling language with threat analysis 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 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 only when all the elements are visited, which expose it to the combinatorial explosion issue in the context of SoS.

3.2.4. Discussion

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

1. Threat analysis approaches only focus on static scenarios to analyze the distribution of threats and hazardous states. They lack guidance and methodologies to support the unknown emergent behavior and do not consider the interactions and security details that trigger the threats and attacks;
2. Graph-based approaches 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 seem to be promising to analyze the SoS interactions and related security issues through simulations.

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

4. The System-of-Systems Security (SoSSec) method to model and analyze secure SoS architectures

To address SoS *modeling* and *analysis* challenges, while considering the SoS specific characteristics and the cascading attacks security emergent behavior, we propose a method, **S**ystem-**o**f-**S**ystem **Sec**urity (SoSSec), devised to answer our main two RQs, as shown in Fig. 1. SoSSec is built following the Model Driven Engineering (MDE) guidelines and introduces several contributions regrouped into two main parts: 1) a Modeling Language called SoSSecML along with its modeling tool to answer our RQ1 and 2) a MAS extension along with the corresponding simulation platform to answer our RQ2, as well as a generator tool to ensure the automatic mapping between the modeling and simulation phases.

4.1. Overview of the SoSSec method

- **Modeling phase**: SoSSecML, a modeling language that allows to model separately the structural, behavioral and security aspects of each CS, guaranteeing the *managerial and operational independence* characteristics, along with the corresponding graphical editor tool. The extension and tool are implemented using MDE mechanisms such as stereotypes, model transformation/code generation and MetaTools;
- **Analysis phase**: We explore agent based approaches and propose an extension of a MAS MetaModel. We implemented this extension in a well-known MAS simulation platform to execute secure SoS architectures in order to discover and analyze emergent unknown security cascading attacks.

Moreover, we complemented SoSSec with a model driven generator tool to (semi)automatically map the secure SoS architecture to executable artifacts. By that, we offer a method with an iterative feature, guarantying an automated mapping (*Mapping Rules* in Fig. 1) between the modeling and analysis phases, thereby allowing the analysis of several secure SoS architectures (iterative aspect) until reaching an acceptable security level.

An earlier and less complete version of the SoSSec Modeling Language, with a proof-of-concept graphical editor implementation from scratch using the Eclipse Modeling Framework (EMF) was presented in Hachem et al. (2016). The EMF implementation lacks Papyrus' advantages of the profile extension mechanism and inheriting SysML concrete syntax to offer a portable, customizable and extensible graphical editor. Moreover, an initial investigation of agent-based simulation was presented in Hachem et al. (2017). However, this work extends the existing works with a MAS security extension and a real-world study domain on which we have applied SoSSec to assess its effectiveness, presenting a unified and complete view of the entire SoSSec method, and focusing on its application to the discovery of a cascading attack on the real-world case study of a smart building.

4.2. SoSSec for secure SoS architecture modeling

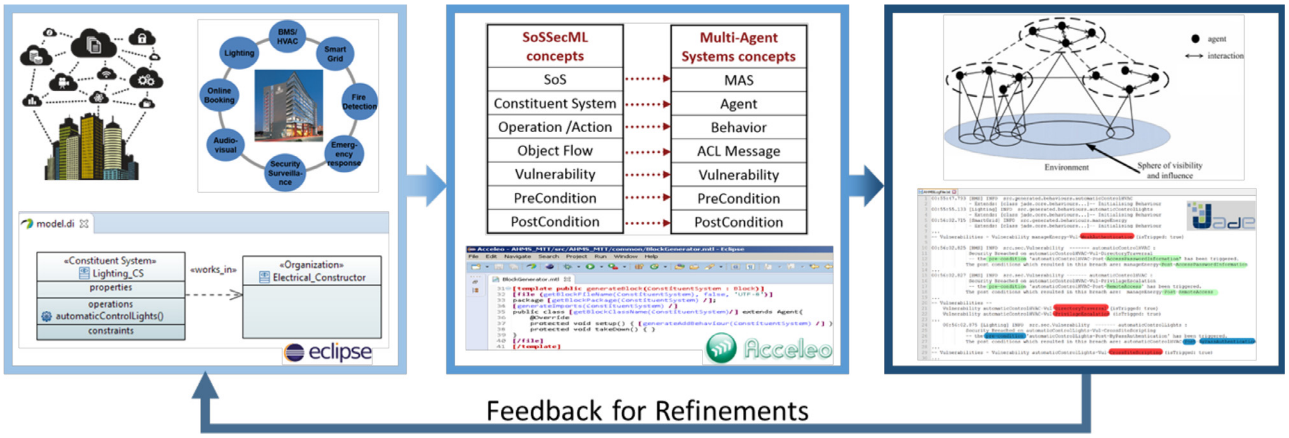
To answer our RQ1, we propose SoSSecML, which is a modeling language that extends SysML. It is defined following the MDE approach whose effectiveness, in the context of SoS, has been argued e.g. in Nakagawa et al. (2013) and Hachem et al. (2016).

To define our SysML extension, and more precisely the SoSSecML abstract syntax/MetaModel, we choose to use the profile mechanism. Indeed, a profile is a lightweight mechanism to extend languages. Its main advantage is the reuse of existing languages abstract and concrete syntaxes and tools instead of starting from scratch. It is used to personalize a modeling language for a specific domain via the stereotypes, tagged values and constraints mechanisms.

SoSSecML (2) extends 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 unknown security emergent behavior.

The main stereotypes that we have defined to model SoS structural, behavioral and security aspects are:

- ***ConstituentSystem*** as a generalization of a *Block*: to represent the CSs as “black boxes” with several *operations* on their interfaces, serving to model the interactions with other CSs. Indeed, what is crucial when modeling a structure of SoS, is to properly describe the CSs boundaries to analyze the regular interactions between CSs as well as emergent behavior (Faldik et al., 2017; Maier, 1998). These CSs participate in



```
[template public generateBlock(ConstituentSystem : Block)]
[file (getBlockFileName(ConstituentSystem), false, 'UTF-8')]
package [getBlockPackage(ConstituentSystem) /];
[generateImports(ConstituentSystem) /]
public class [getBlockClassName(ConstituentSystem)/] extends Agent{
    @Override
    protected void setup() { [generateAddBehaviour(ConstituentSystem) /] }
    protected void takeDown() { }
}
[/file]
[/template]
```

Fig. 1. SoSSec method for secure SoS modeling and analysis.

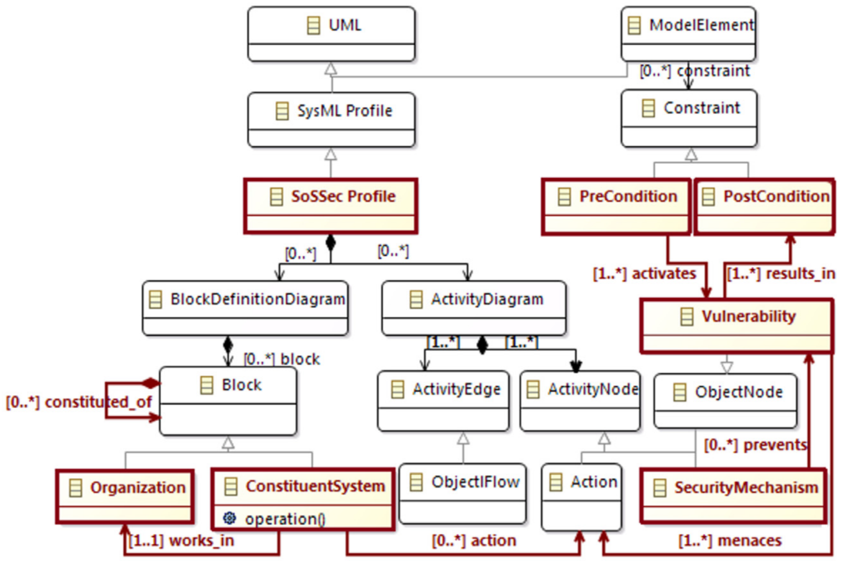


Fig. 2. SysML extension (bold red concepts) for secure SoS architectures modeling. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

the realization of the SoS objective(s) through the *operations* representing the CS global functionalities defined on these interfaces.

- ***Organization*** as a generalization of a *Block*: The CSs in an SoS belong to different organizations/ companies, and each CS may be managed independently by the organization to which it belongs. To model these aspects, we introduced the concept of *Organization* and the relationship *works_in*.
- We reuse the SysML concepts *Action* and *ObjectFlow* to model CS operations and interactions: For each CS, only those operations described as part of 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. Consequently, we consider that the operations of each CS are described in their own activity diagram.
- ***Vulnerability*** as an extension of *ObjectNode*: Since security can be modeled as a behavioral aspect, we enriched the activity diagram with security concepts to support ulterior analysis of security issues. We added the concept *Vulnerability* to represent the back doors or weaknesses in a CS. These vulnerabilities are assigned to the operations on a CS interface, and they could be connected through the CS interactions.
- ***PreCondition*** as a generalization of a *Constraint*: It defines the condition(s) which activates/triggers a vulnerability.
- ***PostCondition*** as a generalization of a *Constraint*: It defines the condition(s) which *results* from an activated/triggered vulnerability.
- ***SecurityMechanism*** as a generalization of an *Action*: To represent possible mechanisms defined to detect, prevent, or recover from a security problem. Security mechanisms could rely on security patterns (Fernandez-Buglioni, 2013) such as: Access control patterns, authentication patterns, encryption and decryption, redundancy, IDS, etc.

4.3. SoSSec for Secure SoS architecture analysis

Having SoS models designed at an early stage gives rise to the necessity of analysis techniques to exploit the models' value and approximate the real system's behavior. Model simulation is one of the most frequently used techniques to analyze SoS successfully (Nielsen et al., 2015). Simulation could cover SoS characteristics like emergent behavior arising from CSs interactions.

Based on the analysis of the related work, we find that Multi-Agent Systems (MAS) are useful techniques to perform SoS simulation due to the similarities between SoS and MAS concepts. MAS are distributed intelligent systems composed of agents that cooperate and coordinate to solve their local goals as well as global goals (Bellifemine et al., 2007).

To answer our RQ2, we present the analysis phase of our SoSSec Method. We propose an extension of MAS to perform analyses of the secure SoS emergent behavior arising from CS interactions.

4.3.1. Alignment of concepts between SoS and MAS

The analysis of the architectures conceived using any modeling language should underly the semantics of the model and its specification features (Medvidovic and Taylor, 2000). The execution (executable artifacts) of the defined architectures should hold the properties described in the models (INC, 2016). Therefore, we performed an analysis of SoS and MAS, and we obtained a number of similarities between their properties:

- Like SoS, MAS are characterized by the absence of global view, by decentralization, by self organization and by large problem solving;
- CS operational and managerial independence could be expressed by the fact that agents in the MAS have their own goals;
- SoS geographical distribution could be translated into the interactions between distributed agents in MAS;
- SoS evolutionary development could be denoted by the flexibility, evolution and openness of MAS;
- Emergent behavior could be expressed by agents autonomy and pro-activity.

Based on these similarities, we aligned the SoS concepts with the corresponding MAS concepts:

- **Constituent System to Agent**
  A CS operates independently to realize its own *individual* goals as well as the SoS *global* goal. It is managerially independent because different CSs could belong to different organizations. Moreover, 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 (operational independence). In MAS, the agents are self-organized and they could be classified in different structural dimensions/organizations that can be distributed over the network (managerial independence).
- **Functionality to Behavior**
  CS operations could describe the CS global functionalities. A CS can interact with several other CSs and execute several operations in parallel to achieve several individual goals. Similarly, a behavior in MAS describes a task that an agent can accomplish. These behaviors can be simple or composite. An agent can execute several behaviors concurrently.
- **Interactions and ACL Messages**
  In the SoS, the CSs are dispersed and they exclusively communicate information (geographic distribution). Consequently, the interactions between different operations defined on different CS interfaces are expressed through the *interaction* concept. Likewise, the agent communications in MAS are achieved through the exchange of *ACL messages*.

4.3.2. MAS MetaModel security extensions

MAS is an appropriate choice for SoS simulation as previously argued, however MAS lack concepts to express SoS security aspects.

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

Nonetheless, existing MAS approaches lack such concepts. For these reasons, we propose an extension of a MAS MetaModel to express SoS security attack related concepts. Our extension (Fig. 3 - bold concepts in red) mainly covers the vulnerability, pre-condition and post-condition concepts detailed in what follows:

- The concept *Vulnerability* represents a weakness that *menaces* an agent behavior. For each behavior we can assign one or many vulnerabilities. A vulnerability *is_triggered* by a matching mechanism.
  As shown in Fig. 4; this mechanism matches the pre-conditions of a vulnerability "i" in the possible cascade with the post-conditions of the previous vulnerabilities in that cascade, triggered through the interactions between the agents, enabling by that the discovery of the possible sequence of vulnerabilities that might be triggered due to the agent interactions. In this way, the interaction that exists between a source CS and a target CS serves as a supplementary condition to trigger a similar condition in the target CS.
- *PreCondition* defines the condition which *activates* a vulnerability.
- *PostCondition* defines the condition which results from an activated vulnerability.

4.3.3. ***Automatic Mappings from secure SoS Architectures to MAS Simulations***

To show the applicability of the MAS security extension that we are proposing, we have implemented this extension in a well-known MAS platform (Hachem et al., 2017), the Java Agent DEvelopment Framework (JADE), to allow the execution/simulation of secure SoS architectures (modeled using SoSSecML) and the discovery of possible related cascading attacks.

We took advantage of MDE mechanisms, in particular the semi-automatic transformation mechanism, to define and implement a model-based code generator tool - a set of Model-To-Text (MTT) transformation rules (Kurtev et al., 2006) - to automatically generate executable MAS artifacts (that could be simulated in our extended MAS platform) from secure SoS architectures designed using SoSSecML.

We defined and applied the transformation rules using Acceleo:[11] an Eclipse-based open source tool. Acceleo uses a template-based approach for the automatic code generation. In such an approach, the mapping between the source model and the produced code is captured through templates-rules. These rules, together with the source model are given as input to a template generator which produces the code in the corresponding output language.

We have implemented a set of MTT rules that ensure the following mappings: *Constituent System* (CS) concept of SoSSecML is mapped to the *agent* class of the JADE platform; the CS functionalities/ operations are mapped to agents behaviors; the flows/ interactions are mapped to*ACL Messages*; and the *vulnerability, pre and post-conditions* of SoSSecML are mapped to the *vulnerability, pre and post-conditions* classes that we introduced to the MAS platform. Moreover, since we are interested only in discovering the sequence of vulnerabilities that are triggered because of the CS interactions, this latter rule is defined in a way to map only the post-conditions of the *triggered* vulnerabilities linked to the *executed* operations. This has a positive

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

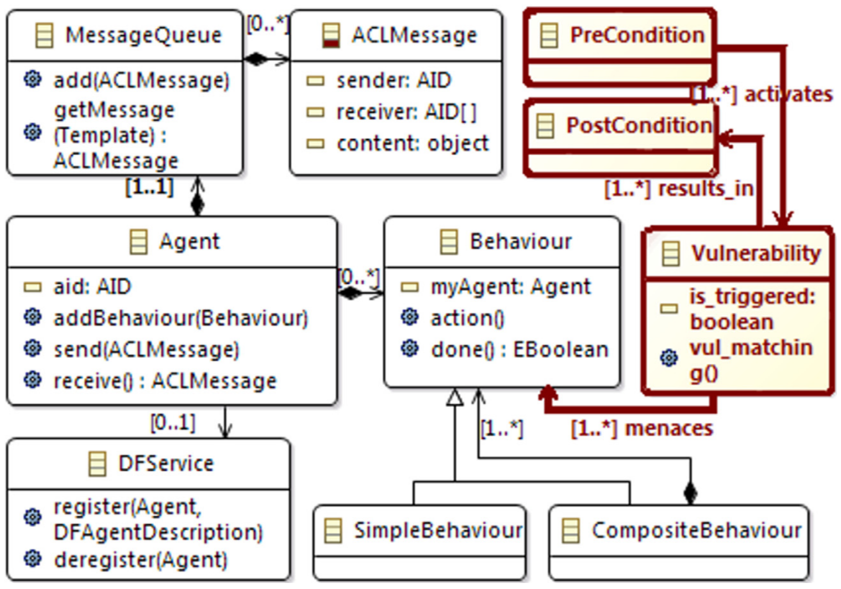


Fig. 3. MAS extension (bold red concepts) for secure SoS architectures analysis. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

```
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 msg
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 Breach
        on 'vul' --'the matched pre-post condition' is_triggered ) }}}
```

Fig. 4. Vulnerability matching mechanism.

impact on performance, eliminating the risk of combinatorial explosion.

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

Having the log file issued from the simulation of the secure SoS architecture enables the security analyst to identify the triggered vulnerabilities, the matching pre-post conditions that triggered the vulnerabilities, as well as the CSs and interactions to which these vulnerabilities belong. Using this information, the security experts are able to analyze the real behavior of the described architectures, as well as to interpret the discovered emergent sequence of vulnerabilities triggered and connected through the CS interactions and as a consequence of validated pre-conditions.

5. The Adelaide Health and Medical School (AHMS) smart building

In Shaw (2002), the author considers *real examples* as one of the validation techniques that could be used as an evidence to demonstrate the validity of a result. Hence, we present in this section a real-world smart building as a study domain, on which we have applied our SoSSec method to model and analyze a secure architecture of the smart building SoS.

It is estimated that by 2050, 70% of the world's population will be living in cities (UN and affairs, 2014). The city of Adelaide, South Australia, is building many initiatives, projects and partnerships with the aim of making Adelaide a safe smart city (Babar, 2016). The *smart campus master plan 2016–2035 project*[12] is one of the projects in this context. The project consists of constructing several smart buildings. A first step on the road of the construction of

[12] https://www.adelaide.edu.au/infrastructure/projects/current/masterplan/.

Table 1
AHMS smart building description.

Constituent System	Operations on the CS interface	**Organization**[a]
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 CO2 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

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

these buildings is the AHMS[13] real-world building of the University of Adelaide. This building, opened for teaching in 2017, costs AU$246 million and is considered the largest capital works' project in the history of the University.

5.1. The AHMS smart building as an SoS

With the increase of their 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 several 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 AHMS is one of these buildings.

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 as described in the protocol presented in Section 6; we elicited the CSs of the AHMS SoS, the organization that govern each CS and the main operations on the CS interfaces employed during the SoS interactions and needed for the achievement of the AHMS global goal, as presented in Table 1.

5.2. The 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. Smart buildings are one of the SoS application domains where security is crucial and cascading attacks may have destructive results. Indeed, smart building infrastructures and services are changing due to their CSs 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 Distributed Denial of Services (DDoS) attacks. Smart buildings' *confidentiality* could be disrupted by unauthorized access to different CSs to reveal personal data related to residents. A building's *integrity* could be compromised if an attacker can control the critical automated CSs of the building and disrupt the operations of these CSs through the injection of malware. The *availability* of a smart building services could be troubled by preventing the delivery of the building CS functionalities. Furthermore, many other attacks may target the building automated CSs,[14] 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 2 shows recent attack examples extracted from different smart building/home attack news.These challenges should be taken into consideration when conceiving the smart buildings solutions.

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 (Garcas et al., 2017) 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 as an important challenge to be addressed. Moreover, a survey[15] carried out in 2017 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 the initial architecture 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.

6. Predicting the cascading attacks for the AHMS smart building using the SoSSec method

In Brereton et al. (2008), the authors assume that using, according to a well-specified protocol, a new method to implement an existing system, can be considered as a validation activity and provides useful information about the advantages and limitations of the new method. Therefore, to apply our method on the AHMS study domain, we followed the planning protocol template pre-

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

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

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

Table 2
Smart buildings/homes attack news and related security concerns.

Attack	Targeted CS	Results
Ransomware attack on a fully booked hotel in Austrian Alps[a], 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.
DDoS attack taking down the central heating system during winter in Finland[b], 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.
DDoS against the Target Store in USA[c], 2014	Building management system (BMS)	Unauthorized access to more than 100,000 devices revealing customers data including 53 million emails and credit card information.
Google's Australian office building hack[d], 2013	Unauthorized access to the BMS	Possible denial of service and false alarms activation.

[a] https://www.nytimes.com/2017/01/30/world/europe/hotel-austria-bitcoin-ransom.html.
[b] http://metropolitan.fi/entry/ddos-attack-halts-heating-in-finland-amidst-winter.
[c] https://securityledger.com/2014/11/third-party-vendor-source-of-breach-at-home-depot/.
[d] http://www.smh.com.au/technology/technology-news/australian-google-building-hacked.html.

sented in Brereton et al. (2008). This protocol improves the reliability of the 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 study domain planning protocol we implemented.

6.1. Background

The background's main aim is to define the research questions to be addressed by the study domain. For our work, after a first meeting with the energy management group of the University of Adelaide, where we presented our SoSSec method and discussed its possible applicability on a smart building SoS, in particular the AHMS, we agreed that the application on this study domain should answer the following questions:

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

6.2. Study domain research design

The central components of a study design are the definition of a study's aim and the identification of the theoretical propositions derived from each RQ.

For our 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 SoSSecML and using its graphical editor;
2. Introduce security knowledge to the AHMS architecture: By defining the possible vulnerabilities related to each AHMS CS 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.

6.3. Study selection

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

- A relatively realistic-scale study domain in the context of smart cities;
- Ability to describe the target system as an SoS;
- 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 possible security concerns from the security team if existing or specify 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 smart city, and considering the fact that designing and implementing a study domain on an entire smart city initiative is currently a rather impractical task, it is more reasonable to design and implement solutions on a smaller but still realistic scale (Babar, 2016). As it fulfills our criteria, a smart building, a part of the smart campus master plan, seems to be a satisfying study domain to apply our SoSSec method.

6.4. Study domain procedures and roles

In order to properly plan the requirements of our study, we describe the details of the data collection plan and the data analysis strategy. We also clarify the roles of the participants in this AHMS study as follows:

- The SoSSec research team is responsible of applying SoSSec to model and analyze the secure software architecture of one or several AHMS scenarios;
- The AHMS team is responsible of offering the needed data as input for the study domain and evaluating the outcome of this study, its validity and usefulness for modeling the future smart buildings of the Adelaide university.

6.5. Data collection

1. To conduct this study, the SoSSec research team collected from the AHMS 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 of the existing publications on security issues or emergent behaviors targeting smart buildings along with the sources of those issues and behaviors.
2. The described data was collected from two sources:
   - Meetings/Interviews: The SoSSec research team had 5 meetings with the AHMS team. In addition, we exchanged with them several emails and documents;
   - Documentation study: Additionally, we studied existing publications such as Plachkinova et al. (2016), Caviglione et al. (2015), Sparrow et al. (2015), W. Sun and Wang (2014), Peacock (2014) and reports/press news such as IntelOrganization (2016), CEICupertinoElectricIncorporation (2015), BBCNewsBritishBroadcastingCorporation (2016), SecurityLedgerIndependentNews (2016) and the sources in Table 2 on the following topics: Smart building systems considering the SoS characteristics; and smart buildings security related concerns and attack news;

6.6. Data analysis

Based on Yin (2003) protocol description, the collected data could be analyzed in three ways:

- Relying on theoretical propositions: The focus on certain data and not other is guided by the theoretical proposition that has formed the study design;
- Thinking about rival explanations: The original theoretical propositions could include some conflicting hypothesis. The collected data could possibly comprise evidences about the possible “other influence”;
- Developing a case description: Define a descriptive framework for organizing the study.

In our work, since the method (language and tools) was already defined, we adopted the first data analysis strategy because it seemed to be appropriate to our situation.

6.7. Schedule

The study was realized in a period of two months.

6.8. Study findings: Secure AHMS smart building architecture modeling and analysis

The SoSSec research team used the SoSSecML modeling language to model the structural and behavioral aspects (including security) of the AHMS smart building SoS. Having the secure AHMS architecture we applied our MTT rules to generate the corresponding code in the extended JADE platform. Afterwards we analyzed the results to discover the possible cascading attacks.

6.8.1. AHMS architecture modeling

AHMS structure modeling. Having collected the data about the functional and security requirements of the AHMS CSs, the SoSSec research team employed the SoSSecML graphical editor to model the structural architecture of the AHMS CSs. Using the *Constituent System* concept of our SoSSecML block diagram, the architects separately modeled each AHMS CS presented in Table 1 and described for each CS the operations on its interface using the block *operation* property. Moreover, the architects represented the company/team that manages each CS using the *Organization* concept and the *works_in* relationship.

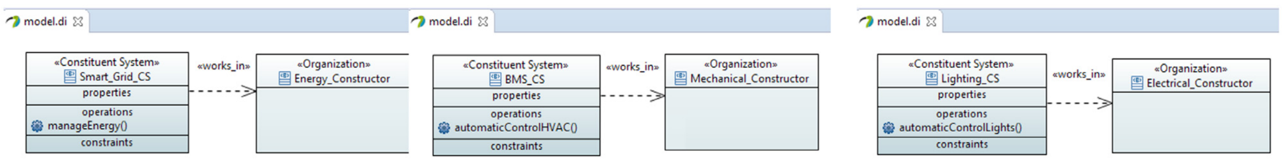
Fig. 5 shows part of the resulting AHMS smart building structural architecture. We can see in Fig. 5a for example, that the stereotype *Constituent System* of our SoSSecML is used to model the *Smart_Grid_CS*. The *operation* property of this extended block is used to represent the *manageEnergy()* operation, 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* organization to which the *Smart_Grid_CS* belongs, this association is represented by the *works_in* relationship. All the other AHMS CSs 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.

AHMS scenario. Among multiple possible scenarios that represent different interactions between the CSs of the AHMS, 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:

1. To reduce the energy consumption in peak hours, the *SmartGrid CS* interacts with the *BMS CS* via the relation “energy_controlling_actions” (Fig. 6). Indeed, the *SmartGrid CS* has as one of its functionalities the **manageEnergy** operation (Fig. 5a) to control the energy use of the *BMS CS* through its operation **automaticControlHVAC** (Fig. 5b);
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 CO2 level)” (Fig. 7) 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 “occupancy_information” (Fig. 8)to reduce the energy consumption;

AHMS security modeling. Since we cannot present the real vulnerabilities of the AHMS CSs for security reasons, we replaced them by some possible realistic vulnerabilities whose definitions are based on 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 as follows: The SoSSec research team studied the existing security publications and attack news to define possible vulnerabilities related to the AHMS CS operations. They extracted the description of the vulnerabilities and their condition details from the Common Vulnerabilities and Exposures (CVE) catalog. CVE is a list of common identifiers for known 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 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:



(a) Smart Grid CS block diagram (b) BMS CS block diagram (c) Lighting CS block diagram

Fig. 5. AHMS CS block diagrams.

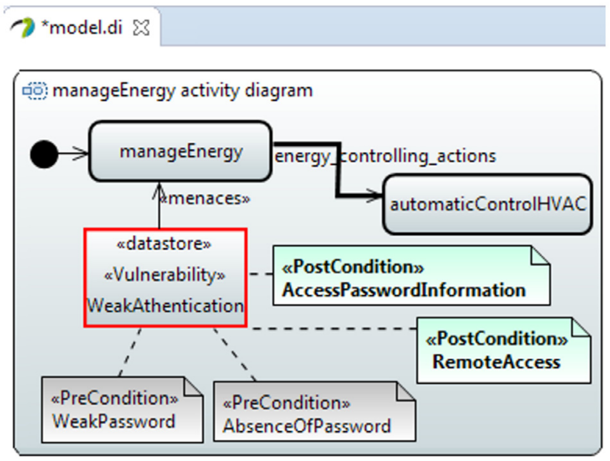


Fig. 6. Smart Grid CS, manageEnergy activity diagram.

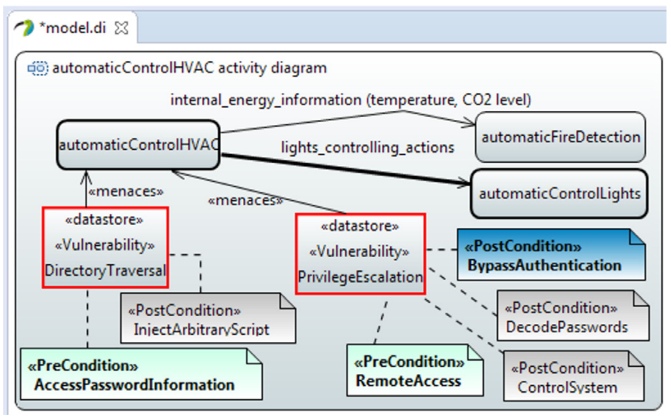


Fig. 7. BMS CS, automaticControlHVAC activity diagram.

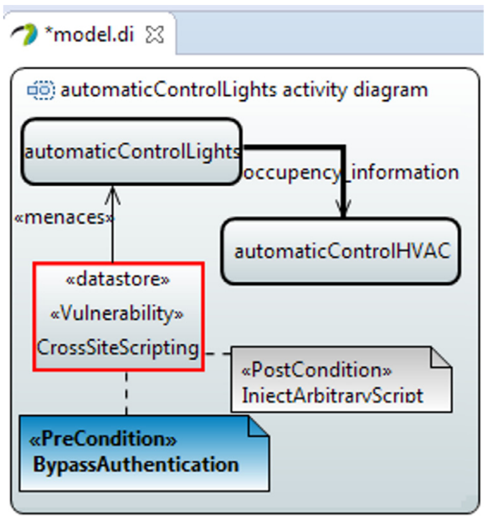


Fig. 8. Lighting CS, automaticControlLights activity diagram.

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** (Fig. 6) - are *menaced* by a *WeakAuthentication* vulnerability 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 (Fig. 7); 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*;
3. For the *Lighting* CS (Fig. 8), we can imagine that attackers 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 DDoS attacks and/or fake emergency alarms.

Table 3 synthesizes the assigned vulnerabilities and their description.

Having the vulnerabilities, the SoSSec research team modeled the 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. To the operations, the architect added 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 Figs. 6–8.

As we can see in Fig. 6; the *manageEnergy* operation defined on the *Smart_Grid_CS* is modeled using the *action* concept,represented in the graphical editor as a rounded rectangle, 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* of the SoSSec MetaModel. To the *manageEnergy* operation, the SoSSec architect links 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

Table 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... 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

the post-conditions *AccessPasswordInformation* and *RemoteAccess* using respectively the *PreCondition* and *PostCondition* stereotypes of the SoSSecML MetaModel. In a similar way, the software and security architect of the SoSSec research team modeled the two other activity diagrams presented in Figs. 7 and 8.

By modeling separate activity diagrams for each CS, we respect the decentralization principle, 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 attribute was respected by considering that the CSs are physically dispersed and only exchange data (not energy or mass). Moreover, by introducing the security concepts, 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.

6.8.2. Analyzing AHMS secure architecture

Having the AHMS SoS secure architecture represented in one model with different views/diagrams, the next step is to simulate this architecture in order to analyze it and detect emergent unknown security cascading attacks. We applied the SoSSec method's analysis phase to discover the possible AHMS cascading attacks. We started the analysis from the elementary minor vulnerabilities we previously modeled, and built upon the interactions specific to the peak hours scenario, that may trigger the vulnerabilities once their pre-conditions are met.

Therefore, the SoSSec team simulation expert applied, on the AHMS models the previously defined MTT rules, to generate the corresponding code. Then they executed this code in our extended JADE platform.

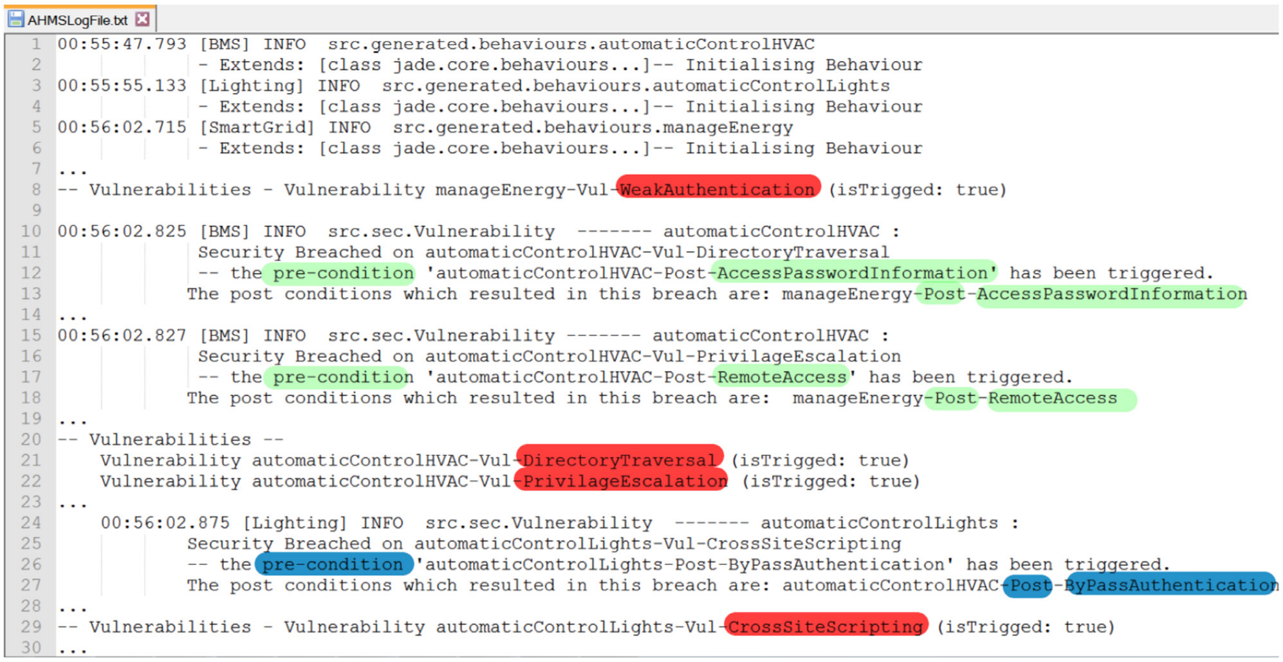
Having the agent interactions and the triggered vulnerabilities captured in the log files, we analyzed qualitatively the security breaches and discovered a cascading attack.

After analyzing the log file, we can notice in Fig. 9; the triggered vulnerabilities resulting from the AHMS agent/CS interactions 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, 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 possibly activated *RemoteAccess* and *AccessPasswordInformation* (Fig. 6). In our extension of the JADE simulation engine, these post-conditions are sent within all the outgoing messages from this agent behavior/CS operation, thus, in our interaction scenario, they are sent to the **automaticControlHVAC**;
2. At the reception, our *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 Fig. 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*;
3. In the same way, the CrossSiteScripting vulnerability of the *automaticControlLights is_triggered* due to the interaction between *automaticControlHVAC* and *automaticControlLights* CS operations, engendered by a matching between the ByPassAuthentication post- and pre-conditions attached respectively to both previously indicated operations (Fig. 8).

From this analysis, the security analyst can conclude that the designed architecture of the AHMS smart building is exposed to an emergent unknown cascading attack (Fig. 10) 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 *InjectArbitraryScript* post-condition of the activated vulnerabilities *DirectoryTraversal* and *CrossSiteScripting* could match the post



```
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. 9. AHMS smart building simulation results - Log file snippet.

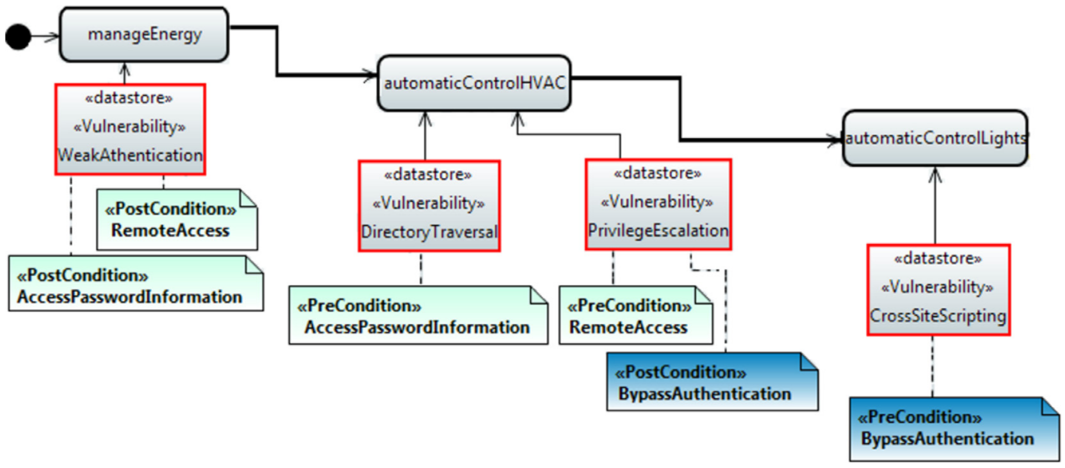

manageEnergy
automaticControlHVAC
automaticControlLights
«datastore» «Vulnerability» WeakAthentication
«PostCondition» RemoteAccess
«PostCondition» AccessPasswordInformation
«datastore» «Vulnerability» DirectoryTraversal
«PreCondition» AccessPasswordInformation
«datastore» «Vulnerability» PrivilegeEscalation
«PreCondition» RemoteAccess
«PostCondition» BypassAuthentication
«datastore» «Vulnerability» CrossSiteScripting
«PreCondition» BypassAuthentication

Fig. 10. AHMS smart building emergent cascading attack.

conditions of other pre-condition vulnerabilities, forming several other possible cascading attacks.

6.8.3. *Discussion of the AHMS secure architecture analysis results*

The discovered cascading attack can lead to a *botnet* opening the door to bigger attacks. Indeed, the *DirectoryTraversal* and *CrossSiteScripting* triggered vulnerabilities of the discovered cascading attack resulted in other situations (cf. Section 5.2) in arbitrary script injections (*InjectArbitraryScript* post-condition), which represent software bots that could lead to a *botnet*. This *botnet* may serve to perform a DDoS attack, steal data, send spam, and/or allow attackers 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 consequences of 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. 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 residents, 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 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 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 so that they propose/model a new architecture alternative that includes effective *security mechanisms*, such as imposing a certain "strongness" of a password to for example address the WeakAuthentication vulnerability, to prevent the discovered cascading attacks.

Furthermore, our method is conceived following the MDE approach, 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 until reaching an acceptable level of security. Indeed, after running a first analysis of this AHMS architecture and discovering the possible cascading attacks to which the architecture is exposed, the architects may decide to include in the architecture some security mechanism to avoid the possible security issues/attacks identified in the SoS architecture. Once the new architecture is modeled, the whole utilization process can be followed again for the new SoS architecture, iteratively, to analyze it.

6.9. Plan to validity

The aim of this study was 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 study to identify the effectiveness of the SoSSec method, its advantages and limitations. The results of the AHMS study domain were discussed in a meeting with the AHMS team that found 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) and for improving their security. These results answered thus the second research question of our study.

6.10. Study limitations

By applying our SoSSec method on the AHMS smart building study domain, we showed the operability of the method and its potential in answering our research questions. However, it is imperative to mention that our study has certain limitations:

- Anonymity: In our study, we excluded some descriptions to protect the anonymity of third parties such as organizations;
- Completeness: The study domain 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, in spite of our 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 study domain CSs, therefore we didn't model them;
- Usability: The application of SoSSec 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 study domain attested the relative ease of use of the method. In addition, the good degree 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 specific analysis of the usability of our SoSSec method should be realized to further improve its ease of use;
- Study selection criteria: The features used to select the AHMS study domain 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);
- Single study: We conducted a single study on one smart building SoS;
- Generalization and scalability: By applying the SoSSec method on the AHMS smart building SoS, without any need of extending it, we assume that it could be generalized to other application domains without many modifications. However, this 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 study domain 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.
- Case evaluation: The study and its results were validated with the AHMS team and by a few additional users (the SoSSec team and four master students), but it was not confronted with security experts. Therefore, the SoSSec evaluation needs to be further enhanced by validating the results with security experts.

However, the 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.

7. Discussion

The main purpose of this research was that the proposed SoSSec method answers the RQs:

In answering RQ1, it enables an appropriate modeling of secure SoS architectures taking into account their specific characteristics and security aspects. In fact, 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 JADE allows the distributed simulation for the execution of the models. Moreover, SoSSec addresses the *emergent behavior* of SoS through the modeling that enables the analysis and discovery/prediction of the cascading attack security emergent behavior. Also, our approach partially covers the *evolutionary development* of the SoS since it is an iterative process that allows the modeling and analysis of several architecture alternatives.

In answering RQ2, SoSSec supports the discovery of unknown cascading attacks arising from the succession of known unresolved vulnerabilities in the independent CSs, that could be connected due to CSs interactions to achieve the SoS global goal. The use of MAS simulation to execute the secure SoS architectures was sufficient to express CSs interactions as described in the architectures. Moreover, the matching mechanism that we defined and implemented, leads to significant results in terms of discovery of high impact attacks such as DDoS and fake emergency attacks. The importance of this mechanism resides in the discovery of the attacks arising only from the enabled vulnerabilities related to the executed operations needed for the realization of the SoS global goal, avoiding by that the combinatorial explosion that results from the analysis of all possible attack paths.

The application of SoSSec on a real-life study domain shows its ability to analyze one of the specific SoS security issues. SoSSec is innovative as there are only few other works that partially address similar RQs (cf. Section 3). However, it is imperative to mention that our method has certain limitations. Firstly, SoSSec does not fully address the emergent behavior characteristic. Being a first step in considering SoS security issues, SoSSec focuses on one kind of emergence resulting from the *known* interactions and vulnerabilities. We plan to extend SoSSec to cover the emergence resulting from *unknown* interactions and vulnerabilities.

Secondly, in this work we modeled the CSs as black boxes. It may be worthwhile to model and analyze the CSs internal cascading attacks and their influence on the SoS security. This is feasible using our SoSSec method since it extends the SysML language and therefore it permits the modeling of each CS structural and behavioral aspects using the basic SysML diagrams (such as the block and activity diagrams), in addition to their security aspects using the SoSSecML security extension. Thirdly, to help architects specify new security mechanisms, the SoSSec should be enhanced with pattern-based recommendations, machine learning methods, content-based filtering approaches or other techniques. The security mechanisms could be incorporated in the design of the SoS architectures using SoSSecML and its graphical editor. Finally, our method with its modeling language and tools needs to be used and validated by industry software and security architects.

8. Conclusion and future work

In this article, we proposed a method called **S**ystems-**o**f-**S**ystems **Sec**urity (SoSSec) to address the challenges of security modeling and analysis during the *architecture phase* of the SoS development life cycle. We followed the guidelines of Model Driven Engineering (MDE) to develop SoSSec consisting of a modeling language, a Multi-Agent Systems (MAS) extension and modeling and simulation tools.

To illustrate our method, we applied it to a real-life smart building, the Adelaide Health and Medical School (AHMS). We modeled the building structural and behavioral aspects, a peak-hours interaction scenario among several CSs, and some of their vulnerabilities together with their pre- and post- conditions (extracted from vulnerability knowledge bases like CVE), using our language, SoSSecML. Then we applied our Model-To-Text transformation rules on the models to generate the corresponding code into the JADE MAS platform that we extended to include our security concepts and other necessary (condition matching) mechanisms. The simulation results were analyzed to discover emerging cascading attacks such as DDoS. Discovering such attacks at the architecture stage of the SoS development life cycle can enable an enterprise to address them at this early phase of the software development lifecycle, and thus saving cost, development time, and protecting the SoS from high impact attacks.

Based on this work, we have identified a number of possible future research directions that can lead to solutions to address challenges of security-sensitive modeling and analysis SoS architectures: (1) Investigating CWE and CAPEC database to enrich SoSSec security models; (2) Refining the security mechanism with design patterns and introducing security metrics that allow comparing quantitatively different secure architecture alternatives; (3) Defining some metrics to quantify the performance of the SoSSec method and the complexity of its vulnerability matching mechanism; (4) Developing the SoSSec method to address the various categories of *emergent behaviors*: In this work, we have 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 a SoS, this list of vulnerabilities will remain *unknown* by other CS, hence, the nomination *unknown known vulnerabilities*. In our future work, we intend to investigate the extension of SoSSec to cover the unknown unknowns emergent behaviors; (5) Extend SoSSec to include other non-functional properties. Whilst this work has focused on security, an interesting investigation will be to extend the SoSSec method to model and analyze other Non-Functional Properties (NFP) together with trade-off analysis methods to compare architectural solutions in terms of several NFP such as the interplay between security and safety or privacy, trust, security and performance.

Acknowledgments

The work of M. Ali Babar was supported in part by the Cyber Security Research Centre Limited, whose activities are partially funded by the Australian Government's Cooperative Research Centres Programme.

References

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

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

Babar, A., 2016. Smart cities: socio-technical innovation for empowering citizens. Aust. Q. 87 (3), 18–25.

Baldwin, W., Sauser, B., Cloutier, R., 2015. Simulation approaches for system of systems: events-based versus agent based modeling. Procedia Comput. Sci. 44, 363–372. Conference on Systems Engineering Research. doi: 10.1016/j.procs.2015.03.032.

BBCNewsBritishBroadcastingCorporation, 2016. Tomorrow's buildings: Help! my building has been hackedhttp://www.bbc.com/news/technology-35746649.

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

Brereton, P., Kitchenham, B., Budgen, D., Li, W., 2008. Using a protocol template for case study planning. In: Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering, pp. 41–48.

Cavalcante, E., Medeiros, A., Batista, T., 2013. Describing cloud applications architectures. In: Proceedings of the 7th European Conference on Software Architecture, pp. 320–323. doi:10.1007/978-3-642-39031-9_29.

Caviglione, L., Lalande, J., Mazurczyk, W., Wendzel, S., 2015. 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 doi:10.1007/978-3-319-20376-8, Los Angeles, United States. https://hal.inria.fr/hal-01182303.

CEICupertinoElectricIncorporation, 2015. Cyber security in smart buildings part 1: disruptive attacks. http://www.cei.com/about-cei/media-room/blog/cyber-security-in-smart-buildings-part-1-disruptive-attacks.

Chiprianov, V., Falkner, K., Gallon, L., Munier, M., 2014. Towards modelling and analysing non-functional properties of systems of systems. In: 9th International Conference on System of Systems Engineering (SOSE), pp. 289–294. doi:10.1109/SYSOSE.2014.6892503.

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

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

Duren, M., Aldridge, H., Abercrombie, R., Sheldon, F., 2013. 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, pp. 48:1–48:3. doi:10.1145/2459976.2460031.

Elahi, G., Yu, E., 2007. A goal oriented approach for modeling and analyzing security trade-offs. In: Proceedings of the 26th International Conference on Conceptual Modeling, pp. 375–390.

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

Fernandez-Buglioni, E., 2013. Security Patterns in Practice: Designing Secure Architectures Using Software Patterns, 1st Wiley Publishing.

Friedenthal, S., Moore, A., Steiner, R., 2014. A Practical Guide to SysML, Third Edition: The Systems Modeling Language, 3rd Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Garcas, L., Ampatzoglou, A., Avgeriou, P., Nakagawa, E., 2017. Quality attributes and quality models for ambient assisted living software systems: a systematic mapping. Inf. Softw. Technol. 82, 121–138. doi:10.1016/j.infsof.2016.10.005.

Guariniello, C., DeLaurentis, D., 2014. Communications, information, and cyber security in systems-of-systems: assessing the impact of attacks through interdependency analysis. Procedia Comput. Sci. 28, 720–727. doi:10.1016/j.procs.2014.03.086. Conference on Systems Engineering Research.

Guessi, M., Neto, V., Bianchi, T., Felizardo, K., Oquendo, F., Nakagawa, E., 2015. 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, pp. 1433–1440. doi:10.1145/2695664.2695795.

Hachem, J.E., 2015. Towards model driven architecture and analysis of system of systems access control. In: IEEE/ACM 37th IEEE International Conference on Software Engineering, Vol. 2, pp. 867–870. doi:10.1109/ICSE.2015.280.

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

Hachem, J.E.L., Pang, Z., Chiprianov, V., Babar, A., Aniorte, P., 2016. Model driven software security architecture of systems-of-systems. In: Proceedings of the 23rd Asia-Pacific Software Engineering Conference (APSEC).
INCOSE Systems Engineering Body of Knowledge, version 1.6. INCOSE UMS. March2016.
IntelOrganization, 2016. Security practices for smart buildings: Good, better, best https://www.intel.com/content/dam/www/public/us/en/documents/solution-briefs/1835security-practices-smart-buildings-brief.pdf.
Jürjens, J., 2002. UMLsec: Extending UML for Secure Systems Development. Springer Berlin Heidelberg, pp. 412–425.
Kimura, D., Osaki, T., Yanoo, K., Izukura, S., Sakaki, H., Kobayashi, A., 2011. Evaluation of IT systems considering characteristics as system of systems. In: 6th International Conference on System of Systems Engineering, pp. 43–48. doi:10.1109/SYSOSE.2011.5966571.
Klein, J., van H., V., 2013. A systematic review of system-of-systems architecture research. In: Proceedings of the 9th International ACM Sigsoft Conference on Quality of Software Architectures, pp. 13–22. doi:10.1145/2465478.2465490.
Kobetski, A., Axelsson, J., 2017. Towards safe and secure systems of systems: challenges and opportunities. In: Proceedings of the Symposium on Applied Computing, pp. 1803–1806. doi:10.1145/3019612.3028252.
Kopetz, H., Höftberger, O., Frömel, B., Brancati, F., Bondavalli, A., 2015. Towards an understanding of emergence in systems-of-systems. In: 10th System of Systems Engineering Conference (SoSE), pp. 214–219. doi:10.1109/SYSOSE.2015.7151925.
Kurtev, I., Bezivin, J., Jouault, F., Valduriez, P., 2006. Model-based DSL frameworks. In: Companion to the 21st ACM SIGPLAN Symposium on Object-oriented Programming Systems, Languages, and Applications, pp. 602–616. doi:10.1145/1176617.1176632.
Lodderstedt, T., Basin, D., Doser, J., 2002. SecureUML: a UML-based modeling language for model-driven security. In: Proceedings of the 5th International Conference on The Unified Modeling Language, pp. 426–441.
Maier, M., 1998. Architecting principles for systems-of-systems. Systems Eng. 1 (4), 267–284. doi:10.1002/(SICI)1520-6858(1998)1:4<267::AID-SYS3>3.0.CO;2-D.
Medvidovic, N., Taylor, R., 2000. A classification and comparison framework for software architecture description languages. IEEE Trans. Softw. Eng. 26 (1), 70–93. doi:10.1109/32.825767.
Meland, P., Paja, E., Gjære, E., Paul, S., Dalpiaz, F., Giorgini, P., 2014. Threat analysis in goal-oriented security requirements modelling. Int. J. Secure Softw.Eng. 5 (2), 1–19. doi:10.4018/ijsse.2014040101.
Merabti, M., Alohali, B., Kifayat, K., 2015. A new key management scheme based on smart grid requirements. In: In Proceedings of the 9th International Conference on Computer Engineering and Applications (CEA), pp. 436–443. doi:10.1109/ICCITECHNOL.2011.5762681.
Merabti, M., Kennedy, M., Hurst, W., 2011. Critical infrastructure protection: a 21st century challenge. In: International Conference on Communications and Information Technology (ICCIT), pp. 1–6. doi:10.1109/ICCITECHNOL.2011.5762681.
Mohsin, M., Anwar, Z., Husari, G., Al-Shaer, E., Rahman, M.A., 2016. IoTSAT: a formal framework for security analysis of the internet of things (IoT). In: IEEE Conference on Communications and Network Security (CNS), pp. 180–188. doi:10.1109/CNS.2016.7860484.
Mori, M., Ceccarelli, A., Lollini, P., Frömel, B., Brancati, F., Bondavalli, A., 2017. Systems-of-systems modeling using a comprehensive viewpoint-based SysML profile. J. Softw. doi:10.1002/smr.1878.
Mori, M., Ceccarelli, A., Zoppi, T., Bondavalli, A., 2016. On the impact of emergent properties on sos security. In: 11th System of Systems Engineering Conference (SoSE), pp. 1–6. doi:10.1109/SYSOSE.2016.7542895.
Muller, G., Dagli, C., 2016. Simulation for a coevolved system-of-systems meta-architecture. In: 11th System of Systems Engineering Conference (SoSE), pp. 1–6.
Muñante, D., Gallon, L., Aniorté, P., 2013. An approach based on model-driven engineering to define security policies using orBAC. In: International Conference on Availability, Reliability and Security, pp. 324–332. doi:10.1109/ARES.2013.44.
Nakagawa, E., Gonçalves, M., Guessi, M., Oliveira, L., Oquendo, F., 2013. 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, pp. 13–20. doi:10.1145/2489850.2489853.
Neto, V., 2016. 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. https://hal.archives-ouvertes.fr/hal-01443187.
Nguyen, P., Ali, S., Yue, T., 2017. Model-based security engineering for cyber-physical systems: a systematic mapping study. Inf. Softw. Technol. 83, 116–135. doi:10.1016/j.infsof.2016.11.004.
Nicklas, J., Mamrot, M., Winzer, P., Lichte, D., Marchlewitz, S., Wolf, K., 2016. 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), pp. 1–6. doi:10.1109/SYSOSE.2016.7542908.
Nielsen, C., Larsen, P., Fitzgerald, J., Woodcock, J., Peleska, J., 2015. Systems of systems engineering: basic concepts, model-based techniques, and research directions. ACM Comput. Surv. 48 (2). doi:10.1145/2794381.
Oquendo, F., 2016. Software Architecture Challenges and Emerging Research in Software-Intensive Systems-of-Systems. Springer International Publishing, pp. 3–21.
Peacock, M., 2014. An analysis of security issues in building automation systems. In: 12th Australian Information Security Management Conference, Perth, Western Australia doi:10.4225/75/57b691dfd9386.
Plachkinova, M., Vo, A., Alluhaidan, A., 2016. Emerging trends in smart home security, privacy, and digital forensics. In: Proceedings of the AMERRICAS conference on information systems (AMCIS).
SecurityLedgerIndependentNews, 2016. Ibm research calls out smart building risks https://securityledger.com/2016/02/ibm-research-calls-out-smart-building-risks/.
Shaw, M., 2002. What makes good research in software engineering? Int. J. Softw. Tools Technol. Trans. 4 (1), 1–7.
Sparrow, R., Adekunle, A., Berry, R., Farnish, R., 2015. 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), pp. 1302–1307. doi:10.1109/MIPRO.2015.7160476.
UN, U.-N. D. o. e., 2014. UN, social affairs, World urbanization prospects.
W. Sun, J.T., Wang, L., 2014. A Smart Home Network Simulation Testbed for Cybersecurity Experimentation, pp. 136–145.
Yin, R.K., 2003. Case Study Research: Design and Methods, Third edition Sage Publications. Italy.
You, G., Sun, X., Sun, M., Wang, J., Chen, Y., 2014. Bibliometric and social network analysis of the SoS field. In: 9th International Conference on System of Systems Engineering (SOSE), pp. 13–18. doi:10.1109/SYSOSE.2014.6892456.
Zhang, L., 2015. Applying System of Systems Engineering Approach to Build Complex Cyber Physical Systems. Springer International Publishing, pp. 621–628.

Jamal EL Hachem is a young Assistant Professor of Software Engineering and Security at the LIUPPA Laboratory of the University of Pau, France. Her research topics cover among others : Software Architecture, Model Driven Engineering, Multi-Agent Systems and Cybersecurity. She published in several international conferences such as ICECCS17, ECSA16, APSEC16, ICSEDS15, SoSE18, etc. She served as a Program Committees member of several national and international workshops and conferences such as SoSE, SISoS, SoSI, WASHES. She also participated in the organization of international conferences such as ICSE15, MODELS16, ECSA16.

Vanea Chiprianov [B.Sc., Politehnica University of Timisoara, Romania, 2007; Ph.D. Telecom Bretagne, France, 2012] is a Senior Lecturer at The University of Bretagne Sud, France. Previously, he was a Lecturer at the University of Pau, France and a Post-doctoral Researcher at the University of Adelaide, Australia. His research interests include software architecture, model driven engineering, security, performance, systems of systems and computer science education. He has published more than 40 research papers in venues such as the Journal of Systems and Software (JSS), the International Journal on Software and Systems Modeling (SoSyM), the European Conference on Software Architecture (ECSA) and the International Conference on Engineering of Complex Computer Systems (ICECCS). He served as reviewer for several international journals and conferences such as JSS, SoSyM and the ACM Special Interest Group on Computer Science Education Conference (SIGCSE).

M. Ali Babar received the M.S. degree in psychology from the Government College University, Lahore, in 1994, the M.Sc. degree in computer sciences from the University of Technology Sydney, in 1999, and the Ph.D. degree in computer science and engineering from the University of New South Wales, in 2006. He is currently a Professor and the Chair of Software Engineering, School of Computer Science, The University of Adelaide, Australia. Prior to this, he was an Associate Professor (Reader) in software engineering with Lancaster University, U.K. Previously, he has worked as a Researcher and project leaders in different research centers in Ireland and Australia. His research projects have attracted funding from various agencies in Denmark, U.K., Ireland, and Australia. He has authored or coauthored more than 150 peer-reviewed research papers in premier software engineering journals and conferences such as the ACM Transactions on Software Engineering and Methods (TOSEM), the IEEE Software, and ICSE. He has recently co-edited a book on Agile Architecting published by Morgan Kaufmann, Elsevier. He is a member of the steering committees of several international software engineering and architecture conferences such as WICSA, ECSA, and ICGSE. He regularly runs tutorials and gives talks on topics related to cloud computing, software architecture and empirical approaches at various international conferences. More information on Prof. M. Ali Babar can be found at http://malibabar.wordpress.com.