Chapter 2

Architectural Description of Systems-of-Information Systems

Milena Guessi, Valdemar Vicente Graciano-Neto, and Elisa Yumi Nakagawa

Abstract

Systems-of-Information Systems (SoIS) refer to collections of software systems that became able to deliver innovative functionalities as well as meet new business opportunities due to novel connections formed with information systems. The software architecture of SoIS plays an important role in their success since they are frequently associated with essential quality attributes, such as sustainability, performance, and reliability. Aiming to study the software architecture of SoIS, additional types of models can be explored for predicting its runtime behavior and structure. Through examples with formal and semi-formal notations, we discuss the advantages and limitations of selected techniques for the description of high level models of SoIS, namely SysML, SosADL, and DEVS. Finally, we outline directions for future work on tailored architectural models for practitioners and researchers dealing with SoIS analysis and design.

2.1. Introduction

Systems-of-Information Systems (SoIS) can be considered as a category of Systems-of-Systems (SoS) that also comprise one or more information systems (IS) (Graciano Neto, Oquendo, & Nakagawa, 2017), which in turn are software systems that collect (or retrieve), process, store, and distribute information (Laudon & Laudon, 2015; Tomicic-Pupek, Dobrovic, & Furjan, 2012). In this sense, other types of systems can be constituents of a SoIS, such as drones, cyber-physical systems, and other SoS. The main purpose of a SoIS is to support novel connections among its constituent systems so as to accomplish one or more business objectives, which can be materialized as flexible and inter-organizational business processes crosscutting a subset of constituent systems that are not exclusively IS (Majd, Marie-Hélène, & Alok, 2015; Saleh & Abel, 2015; Graciano Neto, Cavalcante, Hachem, & Santos, 2017).

The *architecture description* encompasses the set of artifacts used for documenting software architectures (ISO/IEC/IEEE 42010, 2011). As an abstraction of the system,

one or more models can be selected to support different tasks throughout system's analysis and design. For instance, models can be useful for explaining complex parts of the design to non-technical stakeholders, thus contributing for knowledge dissemination among stakeholders (Farenhorst & Boer, 2009). Moreover, models can also be useful for assessing the suitability of architectural decisions since an early stage of the system's life cycle (Capilla, Nakagawa, Zdun, & Carrillo, 2017). Given the complexity of each project, different viewpoints can be selected that will focus the attention of the reader to a single aspect or portion of the system. Therefore, it is important to carefully plan which models will be created so as to promote the readability and changeability of the architectural description without compromising the project's timeline.

The openness and dynamism that are intrinsic to systems-of-systems present additional challenges for researchers and practitioners dealing with their description. As part of a SoS, a constituent system can operate and evolve independently from other constituents, possibly fulfilling parallel tasks whilst being part of the SoS (Maier, 1998). Specially if constituents' operation is concealed from other systems or the SoS itself, architects will need tailored means for ensuring that constituents behave as expected and that certain properties are preserved (Fitzgerald, Bryans, & Payne, 2012). Connections that are formed when a constituent is added to the SoS play an important role in the definition of interactions that can take place at run-time, such as negotiation, orchestration, or choreography. For this reason, we often refer to connections among constituents as mediators (Issarny & Bennaceur, 2013), which are considered as first class entities of SoS architectures.

In this scenario, architects dealing with the description of SoIS will need to select architectural models that will help them to make informed design decisions since an early stage of the system's life cycle. These selections can indicate the need for novel architectural models aiming to better understand SoS dynamic evolution, interface mismatches, and quality characteristics at a higher abstraction level (Batista, 2013). While a literature review reported several languages for describing SoS software architectures (Guessi, Graciano Neto, et al., 2015), there is still no consensus on which models or notations would be more appropriate for this task. Thus, the first step to create tangible artifacts for describing SoIS architectures is to understand the advantages and limitations of specific models and notations.

This chapter investigates the process of creating architectural descriptions for SoIS. First, Section 2.2 presents the main characteristics of SoIS, pointing out key aspects of their design that are important for researchers and practitioners. Then, Section 2.3 introduces architectural descriptions of SoIS, discussing selected viewpoints and notations that can be useful for communication and evaluation. In Section 2.4, we further describe three instances of SoIS, examining how different models have supported in their analysis and design. Finally, in Section 2.5, we outline future directions for tailoring notations and architectural models to the particular needs of SoIS stakeholders.

2.2. Characterization of Systems-of-Information Systems

As a SoS, a SoIS shares the following well-defined characteristics (Maier, 1998): (i) constituent systems retain their *operational independence*, performing other tasks indepen-

dently from the SoS; (ii) constituent systems retain their *managerial independence* so that they can be owned by organizations other than the one running the SoS; (iii) constituent systems are *distributed*, exchanging relevant information with other systems in order to achieve the SoS mission; (iv) the SoS can be *evolutionary developed* in response to individual or collective changes in its constituent systems; and (v) the SoS presents *emergent behaviors* that are the result of new interactions between its constituent systems. Remarkable instances of SoIS encompass smart cities (Graciano Neto, Paes, et al., 2017), space systems (Graciano Neto, Manzano, Rohling, Volpato, & Nakagawa, 2018), and military systems (Paes, Graciano Neto, Moreira, & Nakagawa, 2019). These examples are further discussed in Section 2.4 to exemplify different notations.

Modeling SoS goals requires the identification of global objectives and how individual constituent systems can be assigned to these goals in order to achieve the SoS mission (Silva, Cavalcante, & Batista, 2017). Thus, new constituent systems allow the SoS to offer an unique range of functionality that could not be offered by any of its constituents alone (Maier, 1998). In the context of SoIS, a constituent system contributes with a set of capabilities that can be exploited for achieving specific goals, i.e., activities partitioned into smaller operational tasks that can be distributed to IS constituents matching these capabilities (Graciano Neto, Horita, et al., 2018). Due to SoIS business oriented nature, an internal business process defining the sequence and interdependence between a set of well defined activities is required to govern the SoS operation and the exchange of information between constituents (Graciano Neto et al., 2017). Therefore, the particular characteristics of SoIS brings additional implications to their design, such as (Saleh & Abel, 2015): (i) addressing information and knowledge exchange between IS; (ii) controlling the impact of interrelationships between SoIS that have other SoIS as constituents; (iii) taking responsibility for the information that is generated by the SoIS; and (iv) addressing information interoperability as a key concern.

In this scenario, a shared notation for the specification of SoIS goals must enable the representation of business processes, specially since constituents can be owned by other organizations which can change over time as new constituents join the SoIS (Graciano Neto, Horita, et al., 2018). For instance, mKAOS is a pioneering language specially designed to support missions modeling for SoS and can be considered the state-of-the-art notation for that purpose (Silva et al., 2017). Therefore, SoIS architectures must also be designed to cope with adverse run-time scenarios given that constituents are not necessarily known at design time. Finally, SoIS architectures must also address the interoperability between constituents at a higher abstraction level, specially considering the operational and managerial independence of constituents.

2.3. Architecture Descriptions

Software architectures comprise the fundamental structure of a software system, its elements, the relationship with other elements and to the environment, and the principles governing its design and evolution over time (Bass, Clements, & Kazman, 2012; ISO/-IEC/IEEE 42010, 2011). As tangible artifacts expressing software architectures, architecture descriptions provide concrete ways for accessing systems qualities, sharing architectural knowledge, and preventing software systems' decay (Clements et al., 2011; Kruchten, 2009).

To disseminate best practices regarding the content of such artifacts, the ISO/IEC/-IEEE 42010 (2011) standard establishes the main elements that compose an architecture description. For instance, it establishes that an architecture description should present a set of architecture views, each of which showing the software architecture from a specific architecture viewpoint. Architecture viewpoints select which languages, notations, methods, and model kinds can be used for creating, interpreting, and using a view. In this sense, a model kind is a broad concept encompassing diverse techniques used in architectural models, such as metamodels, templates, languages, and operations. Examples of model kinds include class diagrams, Petri nets, charts, among others.

Complex elements such as architecture frameworks, architecture styles, and Architecture Description Languages (ADLs), are also defined by the ISO/IEC/IEEE 42010 (2011) standard. An architecture framework establishes a common practice for creating, interpreting, analyzing, and using architecture descriptions. To this end, frameworks such as the "4+1" Views (Kruchten, 1995) and Views & Beyond (Clements et al., 2011) select which viewpoints, model kinds, stakeholders, and concerns should be framed by architecture descriptions targeting a particular domain or stakeholder community. In turn, an architecture style selects common model kinds and concerns for a particular class of software architectures. The term notation is used interchangeably in this work as a reference to ADLs, defined by the ISO/IEC/IEEE 42010 (2011) standard as any form of expression used in architecture descriptions. The main difference between programming languages and ADLs is that the latter enables the representation of software systems at a higher abstraction level, i.e., in terms of components, connectors, and configurations (Medvidovic & Taylor, 2000). Nonetheless, ADLs can also target a particular domain or set of concerns. In this regard, it can also be referred to as a domain specific language (Nielsen, Larsen, Fitzgerald, Woodcock, & Peleska, 2015).

To date, there are 124 known languages[1] for the description of software architectures in academia and industry. In this scenario, it is useful to distinguish ADLs based on formalism level, namely (Bass et al., 2012): (i) *formal*, comprising notations with precise (often mathematically based) syntax and semantics; (ii) *semi-formal*, comprising notations with defined syntax but lack of defined semantics; and (iii) *informal*, comprising free style models with ad-hoc syntax and/or semantics. In this scenario, UML is a semi-formal language since new profiles can define a new semantics for existing elements of the language.

The choice for a given notation can be justified based on its suitability for the intended use of architecture descriptions. For instance, practitioners can prefer ADLs that support the specification of both functional and non-functional properties, have a defined semantics to support automated analyses, and both graphical and textual representations for easing the communication among stakeholders (Lago, Malavolta, Muccini, Pelliccione, & Tang, 2015). Correspondences such as refinements, constraints, and dependencies can be defined to enforce consistency between elements of the architecture description. In particular, known inconsistencies should also be recorded by the architect so as to be properly addressed in subsequent stages of the system design. Therefore, it is important to keep record in the description about the rationale behind the selection

[1] Architectural description languages, http://www.di.univaq.it/malavolta/al/

and/or creation of specific views as well as design alternatives and decisions that shaped the resulting software architecture.

In this context, the customization of architecture descriptions for a particular project or stakeholder community is in line with recommended best practices. In a previous study, we investigated the main building blocks of three notations for the description of SoS architectures (Guessi, Cavalcante, & Oliveira, 2015). While we could see similar building blocks in ADLs for SoS and monolithic systems, we noticed specific challenges rising from SoS dynamism and openness. For instance, the architectural configuration of an SoS that we also refer to as a *coalition* must represent the connections between constituents and mediators. However, the coalition presents a combined behavior that is greater than the merely sum of its parts, and this behavior can only be observed at runtime. Moreover, the operational and managerial independence of constituents requires requires a dynamic response of the SoS, e.g., reorganizing its architectural configuration with the remaining constituents and/or creating new mediators to incorporate new ones. Therefore, it is important to take into account these characteristics of SoS when selecting architectural viewpoints and notations for SoIS. Following, we discuss the advantages and limitations of selected viewpoints and notations in light of SoIS characteristics.

2.3.1. Selected Notations and Viewpoits

Several ADLs have been used to document and evaluate SoS (Guessi, Graciano Neto, et al., 2015; Klein & van Vliet, 2013), including semi-formal and formal languages, such as UML[2], SysML[3], and CML (Woodcock et al., 2012). UML and SysML are general purpose languages, the latter is considered an extension of the former. Both languages have been used in SoS descriptions (e.g., Mittal and Risco Martin (2013), Bryans, Payne, Holt, and Perry (2013), Andrews, Payne, Romanovsky, Didier, and Mota (2013), Dahmann et al. (2017)), despite missing first class constructs for mediators and coalitions of SoS architectures (Guessi, Cavalcante, & Oliveira, 2015). For instance, the block definition diagram of the SysML can represent multiple systems. Nonetheless, it does not support representing dynamic properties, such as emergent behaviors, since it is a static model (Guessi, Graciano Neto, et al., 2015). CML is a formal language especially conceived for SoS formal specification, focusing on the verification of emergent behaviors but not on their validation (Fitzgerald, Foster, Ingram, Larsen, & Woodcock, 2013).

Other languages have been used for describing SoS, for instance, the Composable Adaptive Software Systems (COMPASS) (Gokhale et al., 2008) that defines a modeling paradigm for deploying SoS via mediators. This language offers support to modeling constituents, validating the syntax, semantics, and compatibility of assembled constituents, and generating meta-data descriptors that can be used for middleware purposes. Cook, Drusinksy, and Shing (2007) use MSC Assertions, a formal-language extension of the UML sequence diagram superimposed with UML statecharts, to validate SoS run-time behaviors. Griendling and Mavris (2011) use UML in a methodology coined Architecture-based Technology Evaluation. They identify limitations of UML for representing executable models that would suggest the use of Discrete Event notations instead. The

[2]UML, http://www.uml.org/
[3]SysML, http://sysml.org/

Capability Tradeoff (ARCHITECT) UPDM is a formal language that provides UPDM (Unified Profile for DoDAF and MODAF) as a novel, consistent way for creating DoDAF 1.5 and MODAF 1.2 descriptions in UML-based tools, thus offering a compatible way for interchanging descriptions expressed in any of the two notations (Hause, 2010b, 2010a). Finally, bi-graph models have also been used as a formal language for representing SoS in that nodes represent constituents and edges represent communication links between them (Wachholder & Stary, 2015; Gassara, Bouassida, & Jmaiel, 2017; Gassara, Rodriguez, Jmaiel, & Drira, 2017). This notation supports the description of structural properties, which are materialized by links (i.e. system connectivity), and constituents' behaviors, materialized by reaction rules.

In this scenario, we observe that languages for SoS should offer support for (Guessi, Cavalcante, & Oliveira, 2015): (i) partial descriptions of constituents, which are not necessarily known at design time; (ii) environmental modeling; and (iii) dynamic architectures. Aiming to overcome some of limitations found with present ADLs for the description of SoS, we observe several works investigating the transformation of architectural models expressed in ADLs, such as π-ADL, SySML, HLA, or DoDAF[4], into programming and/or simulation languages (e.g., Go language and Simulink[5]). Therefore, we notice that SoS languages also need to bridge the gap between high level architecture descriptions and low level simulation models.

The multitude of languages that can be used for expressing SoS architectures poses new challenges for creating architecture descriptions given that (Guessi, Cavalcante, & Oliveira, 2015): (i) there is no consensus regarding which languages to use for the description of SoS software architectures; (ii) there is a lack for specific guidelines about the selection of suitable formalism levels; and (iii) there is no consensus regarding the essential features of ADLs for the description of SoS software architectures. In particular, the business oriented nature of SoIS requires a specific viewpoint for capturing information as well as decision flow across constituent systems. This architectural view can use BPMN[6] for describing the flexible and inter-organizational processes that enable to achieve SoS goals. However, it is also important to mention that BPMN still presents limitations for SoIS mission design.

2.3.1.1. SosADL

SosADL is a novel formal ADL for SoS that supports the specification of abstract architectures in which concrete constituents are not necessarily known at design-time and abstract connectors are specified so as to be dynamically realized by the SoS in case there is a need to incorporate new constituents and/or reassemble the ones that remained in the colation (Oquendo, 2016b, 2016a). SosADL offers a well-defined semantics founded in the π-calculus and a textual syntax for the specification of high level architecture models (Oquendo, 2016a). In particular, it enables the description of coalitions, i.e., temporary alliances among constituents that collaborate via mediators (Oquendo, 2016c). Thus, the

[4]US Department of Defense Architecture Framework (2010)
[5]Simulink, http://www.mathworks.com/products/simulink/
[6]BPMN, http://www.bpmn.org/

behavior of coalition can be used for specifying how constituents are allowed to communicate at run-time so as to accomplish a given set of missions. The following section will provide examples for the use of SosADL in SoIS architecture description.

Figure 2.1 shows an excerpt of the main building blocks of the SosADL language, including mediators, coalitions, and systems (i.e., constituent systems). Mediators are first-class elements representing the communication links between constituents (Wiederhold, 1992). Both mediators and systems can be further specified in terms of an internal behavior. The architecture defines policies for assembling abstract types of systems and mediators as coalitions, e.g., enumerating abstract types of constituents that can be part of the SoS and the interactions that can take place among them. Coalitions can be further specified by a behavior, data types, and ports exposed to the environment or other constituents. In this sense, ports (i.e., gates and duties) provide abstractions that support modeling the communication between constituents and mediators. For instance, they can receive stimuli from or act upon conditions observed in the environment, hence enabling the interaction of the SoS with its surrounding environment. Data types can define functions, which in turn can be associated with expressions.

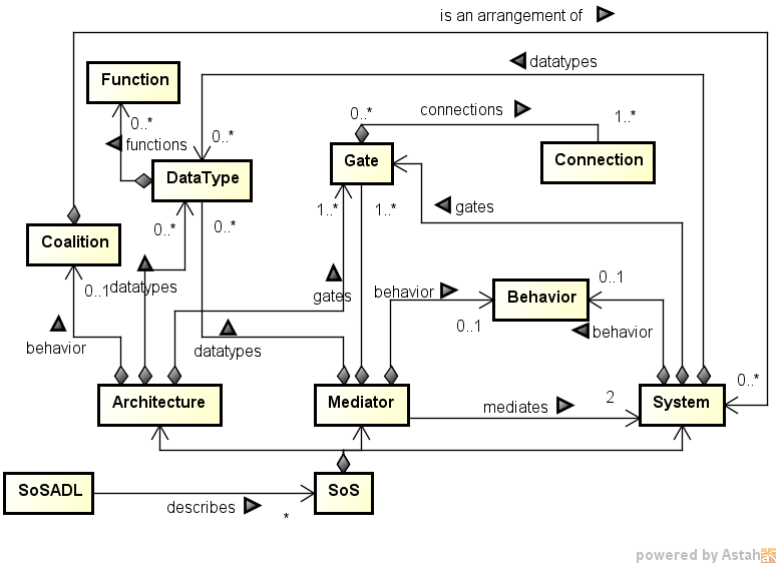

Figure 2.1. An excerpt of SosADL abstract syntax (Graciano Neto, 2016).

2.4. Case Studies on SoIS architecture description

We discuss in this section about the architectural models that have been created for SoIS in three different domains, namely smart cities, space systems, and the military. These models are expressed in SosADL and SysML although other languages have also been used in specific parts of these projects. We explain the rationale for using other languages in the description of each case.

SosADL was used to create models of two different viewpoints: (i) a constituent[7]

[7]We have selected this term instead of Components and Connectors (Clements et al., 2011) due to the fact that this viewpoint also maps the behavior of constituents

viewpoint, which focuses on each individual constituents of a coalition as well as their inner properties (e.g., expected inward and outward connections, behavior, functions, etc.); and (ii) a coalition viewpoint, framing the communication links between constituents and mediators that makes it possible to accomplish the SoS mission. Despite mapping the behavior in both viewpoints, these models are still static, i.e., they are created at design time to explore the run-time behavior of a coalition.

2.4.1. Smart city SoIS

A smart city is a dynamic environment comprising IS that work together to provide efficient and effective services that help to increase the quality of life of citizens (Rech, Pistauer, & Steger, 2018), such as government IS and healthcare IS (Pelliccione et al., 2016). In this scenario, the smart city is a SoIS that potentially involves several constituents that can be, themselves, SoS and SoIS, such as cyber-physical systems-of-systems (CPSoS) (Engell, Paulen, Reniers, Sonntag, & Thompson, 2015; Díaz, Pérez, Pérez, & Garbajosa, 2016), smart houses, smart buildings, smart traffic control, smart grids (for power distribution), smart factories, as well as emergency response systems (Santos, Oliveira, Duran, & Nakagawa, 2015). These constituents are all part of a complex system network, in which systems could compete for shared resources whilst adhering to an overarching, high level business process. Therefore, the collaboration among constituents becomes essential to achieve the SoIS mission, e.g., ensure public security, reduce traffic, monitor imminent flash floods, and manage crisis events.

The authors have been involved in the study of a Flood Monitoring SoS (FMSoS) (Guessi, Oquendo, & Nakagawa, 2016; Graciano Neto, Paes, et al., 2017; Graciano Neto, 2018), one of the constituents of the smart city SoIS that can also relay information to the constituent dealing with emergency and crisis management. FMSoS deploys several sensors to monitor real time conditions of rivers that cross urban areas particularly prone to flooding. At a high abstraction level, FMSoS can be described as having five different types of constituents, namely: (i) *smart sensors*, which are embedded systems distributed throughout the river extension that collect data about water level; (ii) *gateways*, which gather data from other constituents and relay them to external systems; (iii) *crowd sourcing systems*, which are mobile applications used by citizens to warn authorities about the water level in locations outside the range of smart sensors; (iv) *drones*, which are unmanned automated vehicles used by the authority monitoring the water level from the sky, taking pictures of the river and notifying agents of the emergency and crisis management team; and (v) *drone bases*, which are fixed stands for drones' departure and arrival that can be used for recharging their batteries and uploading collected data.

The simulation of the FMSoS is generated from its SosADL description. It includes 42 sensors, nine crowd sourcing systems, and 18 drones. A drone must have its own base, totaling 18 drone bases that transmit collected data through a 3G gateway. There are also 18 gateways distributed along the river bank. In total, there are 20 gateways that can be used to gather data within the coalition and relay these data to external systems (Graciano Neto, Paes, et al., 2017). Listing 2.1 shows an excerpt of the description of a smart sensor in SosADL. The complete description is not shown to improve the readability of this example. From this listing, we can notice that the gate `energy` offers two environment connections, i.e., connections to the external environment, in Lines

11 and 12. The first connection, named `threshold`, reads the minimum energy level required for resuming the sensor operation, and the `power` connection is used to read the battery level of the sensor. Thereby, the description of a connection must specify a name and data type for the information that can be transmitted by the communication channel. Therefore, a SoS architect can use the environment modifier to explicitly determine what constituents, mediators, and coalitions are able to read from and transmit to the environment.

Listing 2.1. A specification of a Sensor in SoSADL.

```
1  //'with' imports declarations suppressed
2  // Description of Sensor as a System Abstraction
3  library WsnSensor is {
4    system Sensor( lps:Coordinate ) is {
5    // Declaration of local types hidden
6    gate measurement is {
7      connection pass is in { MeasureData }
8      environment connection sense is out { MeasureData }
9    }
10   gate energy is {
11     environment connection threshold is in { Energy }
12     environment connection power is in { Energy }
13   }
14   }
15 }
```

Figure 2.2(a) shows the SosADL description of the FMSoS expected behavior, which is named `coalition`. The coalition can be any composition of `sensors`, `gateways`, and `transmitters` (a type of mediator) (Lines 7-9). The `binding` (Lines 11-23) defines how these elements can be connected, which could then enable the warning behavior of the SoIS. The main policy of this behavior states that between each pair of sensors (i.e., `isensor1` and `isensor2`), there must be a transmitter, which in turn will forward data from its input connection `fromSensors` to either the next sensor or to the next gateway by means of its output connection `towardsGateway`, if the transmitter actually mediates a sensor and a gateway. Therefore, the interactions that will take place at run-time can be tailored to the types of elements that actually exist in a concrete coalition.

As previously mentioned, the SosADL description is a static model for the run-time behavior of constituents and coalitions. Therefore, we have implemented an automated model transformation from SosADL to Discrete Event System Specification (DEVS) (Zeigler, Kim, & Praehofer, 2000) aiming to run these models in the MS4ME[8], a simulation environment. Figure 2.2(b) shows an excerpt of a DEVS simulation model for one of the possible scenarios that satisfy the former architecture description. In particular, it shows a coupled model, i.e., a model that defines how constituent systems (which are further described by atomic models) may exchange data with each other, producing an emergent behavior for the coalition as a whole. The DEVS simulation requires a stimuli generator, an artificial entity that has been introduced to the coupled model for autonomously producing stimuli that emulate the SoS environment (Graciano Neto, Paes, et al., 2017; Graciano Neto, 2018). In this example, this system emulates the handover of

[8]MS4ME, http://ms4systems.com/

Figure 2.2. A SosADL coalition description mapped to a DEVS simulation model.

real world data, such as the `lps` coordinates obtained by a GPS and the water level collected by a smart sensor, between sensors towards the gateway. The interactions between constituents and connectors are created by iterating on the set of sensors and transmitters in the coalition, creating one interaction for each element of the unification segments in the SosADL description. Figure 2.3 shows a screenshot of a smaller instance of this simulation running in the MS4ME environment. The simulation supports the observation of emergent behaviors in the coalition, offering an animated view of the architecture description that was originally represented in SosADL. In particular, this simulation can be used to study the roles of individual constituent systems, provided and required interfaces, and data flow across the coalition. Details on the transformation from SosADL to DEVS are further discussed by Graciano Neto, Garcés, et al. (2018).

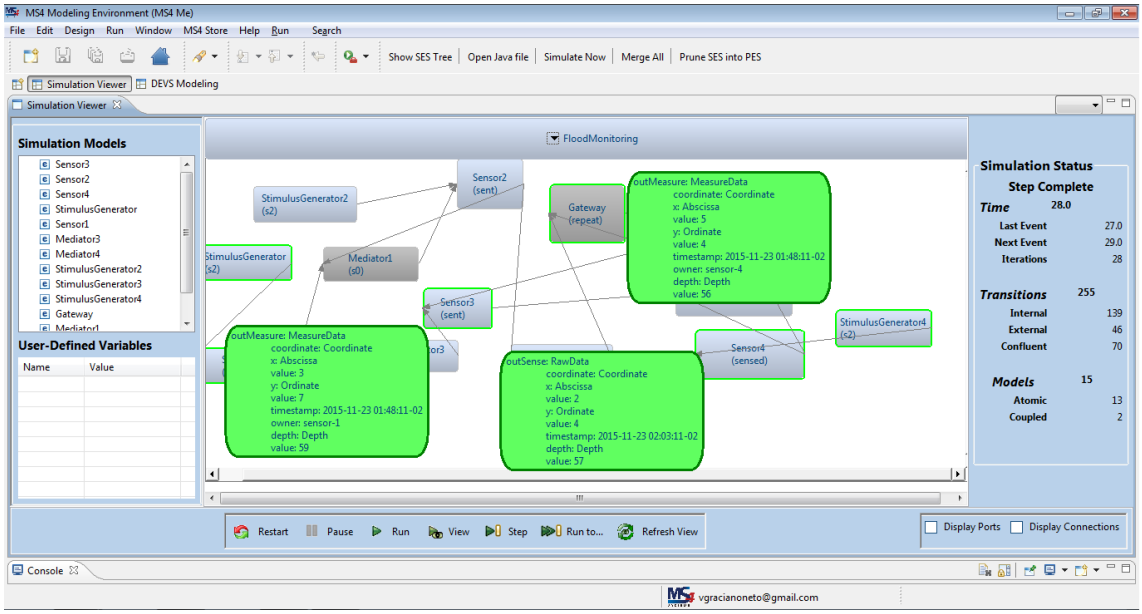

Figure 2.3. Screenshot of the Flood Monitoring SoS model in DEVS running in MS4ME.

2.4.2. Space SoIS

Space systems are indispensable to modern life, finding applications in telecommunications, space, climate, and natural resources monitoring, early warning systems, and national and sea coast surveillance[9] (Graciano Neto, Manzano, et al., 2018). Since a Space SoS also comprises some software-intensive IS, it can also be considered a SoIS. The goals of the Space SoIS are explicitly expressed in terms of a business process, i.e., a well-defined sequence of inter-dependent activities distributed among its constituent systems. Approximately 800 constituents (both on the ground and in the space) can accomplish important goals of this SoIS, such as global telecommunication capabilities, global position services (GPS), weather forecast, and military observation (Yamaguti, Orlando, & Pereira, 2009; Graciano Neto, Horita, et al., 2018; Graciano Neto, Manzano, et al., 2018).

Figure 2.4 illustrates the main constituents of the Space SoIS spread over the Brazilian territory. The Space SoIS is composed of the following different types of constituents (Graciano Neto, Manzano, et al., 2018; Graciano Neto, Horita, et al., 2018) . Satellites are the main constituents of a Space SoIS. Each satellite is divided into several subsystems, having an onboard computer, power system, propulsion system, altitude control, communication system, sensors, infrared cameras, solar panels, batteries, and reaction control system. A satellite establishes contact with systems on the ground when it overflies a ground station, collecting data from constituents such as Data Collection Platforms (DCP) and capturing photographs from strategic locations, such as Amazon forest. Additional goals for a Space SoS may include: (i) monitoring of deforestation and fires; (ii) telecommunications to support Internet and TV; (iii) scientific missions, such as study of solar behavior and planetary exploration; (iv) river monitoring in the event of

[9]Brazilian National Program of Space Activities (2012-2021), https://goo.gl/7h9ETV

environment disasters; and (v) detection of tsunamis and hurricanes. Other constituents illustrated in this figure are:

Figure 2.4. Illustration of a Brazilian Space SoIS for information exchange via satellites (INPE, 2019).

1. **Command and Control Center (C2)**: an IS located in São José dos Campos. It generates goal requests to be accomplished by the SoIS;

2. **Satellite**: a synchronous polar orbit satellite that takes photographs of the Earth every five days.

3. **Ground Station:** an IS located in Cuiabá that receives and handovers data to satellites, temporarily storing data and location of these satellites;

4. **Remote Sensing Data Center:** an IS that receives, stores, processes, and distributes images and data from remote sensors;

5. **Data Collection Platform (DCP)**: a cyber-physical system whose electronic sensors measure environmental variables, such as precipitation, atmospheric pressure, solar radiation, temperature, air humidity, dew point, wind direction and speed, and detects variations in water body level[10]. They are positioned so as to cover the entire Brazilian territory and are of particular necessity to monitor remote regions where no other communication technology is available, such as in the center of the Amazon rainforest. Therefore, these systems can relay collected information to

[10]Simge website, http://www.simge.mg.gov.br/simge/sobre-o-simge

overflying satellites, which in turn will hand over that information to ground stations in their path until finally reaching their final destination and thus fulfilling the Space SoIS goal to monitor remote areas within the national territory.

Goals of the Space SoIS adhere to a well defined business process (Graciano Neto, Manzano, et al., 2018). The process for *acquiring environmental data*, for instance, is composed of many activities performed by different constituent systems (Graciano Neto, Horita, et al., 2018). The SoIS involves governmental institutions and enterprises of the business sector and DCPs owners. The Command and Control constituent is operated by expert users. This constituent supports the elaboration of operational plans and distribute commands to Ground Stations, which configure antennas and rotors, establish links with satellites, and send commands to be performed by the satellites accordingly. Data is captured by DCPs spread all over the country territory and transmitted back to overflying satellites orbiting the Earth.

The process of taking photographs is similar, substituting the task of obtaining data from DCPs by the task of taking photographs with monitoring cameras and other sensor devices. These processes are inherently flexible and inter-organizational since many institutions that own constituents cooperate with each other to fulfill the SoIS goal. As new DCPs, satellites, and IS may join this SoIS at run-time, the Space SoIS architecture is inherently dynamic, even if changes to its constituents do not occur frequently. Indeed, there is usually a fixed set of systems (including IS) that contribute for achieving the SoIS mission, guided by a business process. Despite other types of systems involved (such as satellites and DCPs), ISs that compose the SoIS also follow their own business processes, serving SoIS missions on demand.

The second author has modeled the Space SoIS architecture with a C2 center, a data center, a ground station, six satellites, and 249 DCP stations (totaling 258 constituents). The orbits were defined according to a study on the constellation of satellites (Carvalho, Santos Lima, Santos Jotha, & Aquino, 2013). Maximum and average contact times, maximum and average revisit times, percentage of satisfactory revisits, and the average number of contacts per day were recorded during the preparation for simulation (Manzano, Graciano Neto, & Nakagawa, 2019). The complete architectural description of this coalition in SosADL and DEVS is externally available[11]. Following the same rationale for the model in Listing 2.1, Listing 2.2 shows a simplified version of an architectural description in SosADL of one of the satellites involved in the Brazilian Space SoIS. The Satellite Amazonia handles several types of data. As it can be seen, the satellite can manage `Images` that represent photographs captured from the Brazilian territory (Line 2), `Telecommands` (Lines 5-7), which are instructions/assignments of activities that are received from ground experts to determine satellite operation, and many other types of data (Lines 8-18). The surrounding environment of the satellite is also modelled and monitored via external gates and their respective connections that sense the environment (Lines 21-26). Other types of gates also exist to capture images via camera (Lines 28-30), send data to the ground station (Lines 32-34), offer GPS services and to locate itself in the space (Lines 36-38), and its internal behavior (Lines 40-85). Its behavior comprises continually checking battery levels (Lines 47-57). When it is not charging, the satellite may

[11] Space SoIS architecture description, http://www.inf.ufg.br/~valdemarneto/projects/spacesos.html

open solar panels to recharge[12]; wait for instructions (`telecommands`) from the ground (Lines 59-73); provide GPS location (Lines 75-78); and receive and send images when the ground station is within range and it is overflying a specific location (Lines 75-82).

Listing 2.2. Excerpt of a satellite modelled in SosADL.

```
1  system SatelliteAmazonia3( lps:Coordinate ) is {
2    datatype Image is tuple { name:String, extension:String, content:Binary }
3    datatype CollectedImages is sequence { Image }
4
5    datatype Telecommand is tuple { id:integer, date:Calendar, orbitId:integer,
6      name:String, instruction:integer, coordinateToBeMonitored:Coordinate
7    }
8    datatype Binary
9    datatype Orbit
10   datatype Power
11   datatype SatelliteHeight
12   datatype SatelliteTemperature
13   datatype Latitude is Double
14   datatype Longitude is Double
15   datatype SatellitePosition is tuple { x:Latitude, y:Longitude }
16   datatype Coordinate is tuple { x:Latitude, y:Longitude }
17   datatype Establish
18   datatype Distance
19   ...
20
21   gate satelliteState is {
22     environment connection power is in { Power }
23     environment connection orbit is in { Orbit }
24     environment connection temperature is in { Integer }
25     environment connection height is in { Integer }
26   }
27
28   gate camera is {
29     environment connection image is in { Image }
30   }
31
32   gate telemetry is {
33     connection telemetry is out { Image }
34   }
35
36   gate location is {
37     environment connection coordinateSatellite is in { SatellitePosition } connection
           coordinate is out { SatellitePosition }
38   }
39
40   behavior main is {
41     value telecommand1 : Telecommand = any
42     value powerThreshold : Power = 20 //battery threshold in 20 percent.
43     value image1 : Image = any
44     value powerNow : Power = any
45     value distanceMax:Distance = 5
46
47     repeat{
48       via satelliteState::power receive powerNow
49       if (powerNow > powerThreshold) then {
50         value powerNow = powerNow - 10
51       } else {
52         value powerNow = 100
53       }
54
55       choose {
56         via establish::establishConnectionGS receive establish
57         if(establish = 1) then {
58           via operation::telecommand receive telecommand
59         }
60       } or {
```

[12]This is not specified in this listing.

```
61          via establish::establishConnection receive establish
62          if(establish = 1) then {
63          choose {
64            via notification::terrestrialMeasurereceive terrestrialData
65            do terrestrialDataBuffer::append(terrestrialData)
66          } or {
67            via notification::aquaticMeasure receive aquaticData
68            do aquaticDataDataBuffer::append(aquaticData)
69          }
70        }
71        } or {
72          via location::coordinateSatellite receive satellitePosition
73          via location::coordinateSatellite send satellitePosition
74        }
75        if(distance(satellitePosition, telecommand::coordinateToBeMonitored)
76        <= distanceMax) then {
77          via camera::image receive image1
78          via telemetry::telemetry send image1
79          via camera::image receive image1
80          do collectedImages::append(image1)
81          via telemetry::telemetry send image1
82        }
83     }
84    }
85  }
```

Listing 2.3 shows an excerpt of the description of the Space SoIS in SosADL. Constituents and mediators have all been identified for that architecture (Lines 4-13). In this version, we can observe Data Center, Command and Control, Ground Station, Satellite, some mediators, and one DCP. Bindings (Lines 15-33) show part of the description that determine how output connections can be linked to predefined input connections for pairing constituents in the Space SoIS. Then, DEVS simulation models of this architecture have been created based on this description. In this case, the simulation enabled to visualize goals as inter-organizational and potentially flexible business processes.

Listing 2.3. Excerpt of a Space SoIS architecture modelled in SosADL.

```
1  sos spaceSoSArchitecture is {
2    architecture spaceSoSArchitecture( ) is {
3      behavior coalition is compose {
4        dataCenterCP is DataCenter
5        commandAndControlSJC is CommandAndControl
6        groundStation is GroundStation
7        satellite is SatelliteAmazonia3
8        mediator1 is MediatorDataCenterToC2
9        mediator2 is MediatorC2ToGround
10       mediator3 is MediatorGroundToSatellite
11       mediator4 is MediatorGroundToDataCenter
12       pcd31980 is PCDTerrestre
13       mediatorPcd1 is MediatorPCDtoSatellite
14       }
15       binding
16       {
17         unify one { dataCenterCP::telemetryrequirement::request } to one
18         { mediator1::transmit::request }
19         and
20         unify one { mediator1::transmit::request } to
21         one { commandAndControlSJC::requests::request } and
22
23         unify one { commandAndControlSJC::requests::operation }
24         to one { mediator2::operation } and
25         unify one { mediator2::transmit::operation }
26         to one { groundStation::operation::operation } and
27         ...
28
29         unify one { groundStation::sendData::telemetry }
```

```
30          to one { mediator4::transmit::telemetry } and
31          unify one { mediator4::transmit::telemetry } to
32          one { dataCenterCP::datarequirement::telemetry } and
33          ...
34        }
35    }
36  }
```

2.4.3. SysGAaz

We also investigate the description of a real military SoIS of the Brazilian Navy, named SySGAaz. This SoIS monitors the Blue Amazon, i.e., a region that extends along the Brazilian coast boarder, comprising 350 nautical miles (Paes et al., 2019). The Blue Amazon is equivalent in size to the Amazon rainforest (4.5 million km^2), but concentrates 95% of the Brazilian foreign trade flow and 80% of the Brazilian oil reserves.

Many public organizations participate in the operation of this SySGAaz, including the Federal Regulatory Agency, the Brazilian Army, as well as organizations from the private sector. This SoIS interfaces with all of these entities aiming to minimize information exchange and coordinate missions with the Brazilian Navy. Paes et al. (2019) indicate three essential types of systems of naval operations: (i) Command and Control Naval System (SisNC2); (ii) Operational Intelligence System (SIOp), and (iii) Maritime Traffic Information System (SISTRAM). SisNC2 is a planning, monitoring, and decision support system for the Maritime Operations Theater Commander (TOM). SIOp plans, executes, and controls naval operations at various levels, interacting with others systems to provide information and knowledge to Operational Intelligence Center (CEIOP). Moreover, SisNC2 aggregates regionally collected information from Brazilian Navy units, external organizations, and military organizations to obtain knowledge in specific areas, such as acoustic and electronic warfare, direction finding and maritime traffic control maritime. Finally, SISTRAM handles the maritime traffic control of merchant ships in the Brazilian coast.

External systems of other armed forces branches as well as third party organizations provide input to several scenarios and operating levels in the Brazilian Navy. Some examples are: Operations Planning Information System (SIPLOM), Integrated Border Monitoring System (SISFRON), General Air Synthesis Transmission System (SISTRASAG), and Amazon Protection System (SIPAM). SIPLOM is the main Command and Control (C2) system of Ministry of Defense that displays location, characteristics, and interrelations of constituted means of peace, crisis, and war operations. SISFRON is the system of continuous monitoring of areas of interest, particularly of the border, and integrates existing systems, based on a communications infrastructure, supported by information security. SISTRASAG is a system of the Brazilian Aerospace Defense Command that provides encrypted aerial information (radar images) to authorized military organizations. Finally, SIPAM integrates a comprehensive capability that supports remote weather monitoring and surveillance of the Amazon rainforest.

Paes et al. (2019) use multiple model kinds in the description of the SySGAaz software architecture. In particular, they select the Business Process Model and Notation (BPMN) and Data Flow Diagram (DFD) for representing operational processes, and a combination of UML and SysML models for materializing the coalition and constituent

descriptions. One of the issues found with the use of multiple model kinds in this project was the lack of compatible architectural elements, which could have led to incomplete and inconsistent models across different notations. Therefore, if model kinds are not carefully selected, the use of multiple model kinds can contribute for the lack of standardization in the development phase. Another issue with SysML and UML is that these are mostly descriptive and static notations, missing dynamic constructs that would support the description of emergent behaviors in the coalition. Hence, there is still a need for a graphical ADL that supports a holistic and comprehensive representation of the software architecture of SoS and SoIS, comprising both static and dynamic views of its structure and behavior[13].

2.5. Conclusions and Future Work

This chapter presented our latest developments regarding the architectural description of SoIS. We have experienced different notations for the description of coalitions, behaviors, and properties. We have also created models that allowed us to simulate the surrounding environment of SoIS at the architectural level, which is an important achievement in systems and software engineering (Graciano Neto, Paes, et al., 2017; David et al., 2013; Iqbal, Arcuri, & Briand, 2015; Cheng, Sawyer, Bencomo, & Whittle, 2009). In particular, we showed how static models that were originally expressed with SoSADL were later transformed in DEVS dynamic models and simulated in MS4ME. From these previous experiences, we draw a few lessons for the description of SoIS, specifically:

- A textual ADL, such as SosADL, supports a static view of the structure and behavior of coalitions and constituents of SoIS. This description still needs to be complemented by a dynamic view, which will support the description of emergent behaviors that are anticipated at design time. Simulation models and models at run-time can be used for this purpose, helping to prevent economic losses and/or critical errors that incur from flawed system specifications. Dynamic models can be manually or automatically produced, depending on the feasibility of implementing model transformations. Simulation models can be based on different paradigms, such as discrete events or system dynamics. DEVS is a prominent option for discrete events specification whilst Modelica[14] is an option for system dynamics. In particular, the goal specification is embedded in the SoIS behavior;

- A graphical ADL, such as UML or SysML, supports a static view of the SoIS. These views are expressed by a set of models that capture different facets of the SoIS. Frequently, we can combine two or more graphical languages to overcome the weaknesses of a single notation. For instance, mKAOS (Silva et al., 2017) can be specifically selected for the graphical representation of SoIS missions. Despite known limitations of SysML for representing process flows, block diagrams can be used for representing constituents and their interfaces, sequence diagrams

[13]It is important to distinguish between the pairs structure-behavior and static-dynamic. SosADL offers constructs to represent both structure and behavior of SoS/SoIS, but it is not a dynamic language. On the other hand, DEVS simulation models and models at run-time provide a dynamic view of the SoS/SoIS architectural descriptions.

[14]Modelica, https://www.modelica.org/

for mapping process flows across constituents, and UML diagrams for refining the architecture of each constituent, showing its required and provided interfaces that enable its interaction with other constituents (Graciano Neto, Horita, et al., 2018).

As future work, we identify two research directions for both practitioners and researchers dealing with SoIS architectures design.

- Formalization of SoIS architecture description. In this chapter, we reported our experiences with formal and semi-formal notations for SoIS description. While SysML offers a graphical notation that helps in the communication of architectural knowledge, the lack of formal foundations hampers its applicability for automated analysis. In fact, concealing mismatches across architectural models can be harmful specially if SoS perform a safety-critical mission. In this sense, formal notations will provide better support for automatically propagating changes throughout subsequent stages of the development life cycle (Mens, Magee, & Rumpe, 2010), making them more suitable for dealing with the evolutionary development of SoS and SoIS. Moreover, language developers will need to address disadvantages that usually come from using formal notations, such as requiring high experience level and specialized training, by concealing the use of formal languages from the user.
- Viewpoints for describing SoIS software architectures. In this chapter, we addressed the description of SoIS from a constituent and coalition viewpoints. Nonetheless, other viewpoints could be required, such as the mission viewpoint discussed in this chapter, but there is still no consensus on an architecture framework for SoS (Guessi, Graciano Neto, et al., 2015), which includes SoIS. Thus, depending on which set of viewpoints are needed, ADLs may need to provide support for additional sets of features, such as tailored constructs for business processes.

Acknowledgements

The authors thank Prof. Dr. Adair Rohling (UTFPR) for his work on the Space SoIS, Prof. Dr. Carlos Paes (PUC-SP) for his work on the Navy SoS, and Wallace Manzano (ICMC-USP) for his work on implementation and simulation of DEVS models. This work is supported by the São Paulo Research Foundation (FAPESP), grants 2012/24290-5, 2017/22107-2, and 2017/06195-9, and of the Brazilian National Research Council for Scientific and Technological Development (CNPq), grant 312634/2018-8.

References

Andrews, Z., Payne, R., Romanovsky, A., Didier, A., & Mota, A. (2013). Model-based development of fault tolerant systems of systems. In *7th IEEE International Systems Conference (SysCon 2013)* (pp. 356–363). Orlando, FL, USA: IEEE.

Bass, L., Clements, P., & Kazman, R. (2012). *Software architecture in practice* (3rd). Indianapolis, Indiana, USA: Addison-Wesley Professional.

Batista, T. (2013). Challenges for sos architecture description. In *1st International Workshop on Software Engineering for Systems-of-Systems (SESoS)* (pp. 35–37). Montpellier, France: ACM.

Bryans, J., Payne, R., Holt, J., & Perry, S. (2013). Semi-formal and formal interface specification for system of systems architecture. In *7th IEEE International Systems Conference (SysCon 2013)* (pp. 612–619). Orlando, FL, USA: INCOSE.

Capilla, R., Nakagawa, E. Y., Zdun, U., & Carrillo, C. (2017). Toward architecture knowledge sustainability: Extending system longevity. *IEEE Software*, *34*(2), 108–111.

Carvalho, M. J. M., Santos Lima, J. S., Santos Jotha, L., & Aquino, P. S. (2013). CONASAT: Constellation of Nano Satellites for Environmental Data Collection (in Portuguese). In *16th Brazilian Symposium on Remote Sensing* (pp. 9108–9115). Foz do Iguaçu, Brazil: National institution of Space Research.

Cheng, B. H. C., Sawyer, P., Bencomo, N., & Whittle, J. (2009). A goal-based modeling approach to develop requirements of an adaptive system with environmental uncertainty. In A. Schürr & B. Selic (Eds.), *Model driven engineering languages and systems* (pp. 468–483). Berlin, Heidelberg: Springer Berlin Heidelberg.

Clements, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Little, R., ... Stafford, J. (2011). *Documenting Software Architectures: Views and Beyond* (2nd ed.). Addison-Wesley.

Cook, T. S., Drusinksy, D., & Shing, M. T. (2007). Specification, validation and runtime monitoring of soa based system-of-systems temporal behaviors. In *3rd IEEE International Conference on System of Systems Engineering (SoSE 2007)* (pp. 1–6). San Antonio, USA: IEEE.

Dahmann, J., Markina-Khusid, A., Doren, A., Wheeler, T., Cotter, M., & Kelley, M. (2017). Sysml executable systems of system architecture definition: A working example. In *11th Annual IEEE International Systems Conference (SysCon 2017)* (pp. 1–6). Montreal, Canada: IEEE.

David, O., II, J. A., Lloyd, W., Green, T., Rojas, K., Leavesley, G., & Ahuja, L. (2013). A software engineering perspective on environmental modeling framework design: The object modeling system. *Environmental Modelling & Software*, *39*, 201–213. Thematic Issue on the Future of Integrated Modeling Science and Technology.

Díaz, J., Pérez, J., Pérez, J., & Garbajosa, J. (2016). Conceptualizing a framework for cyber-physical systems of systems development and deployment. In *10th European Conference on Software Architecture Workshops (ECSAW 2016)* (pp. 1–7). Copenhagen, Denmark: ACM. doi:10.1145/2993412.3004852

Engell, S., Paulen, R., Reniers, M. A., Sonntag, C., & Thompson, H. (2015). Core research and innovation areas in cyber-physical systems of systems. In M. R. Mousavi & C. Berger (Eds.), *Cyber physical systems. design, modeling, and evaluation* (pp. 40–55). Cham: Springer International Publishing.

Farenhorst, R. & Boer, R. C. (2009). Knowledge Magament in Software Architecture: State of the Art. In M. A. Babar, T. Dingsøyr, P. Lago, & H. van Vliet (Eds.), *Software architecture knowledge management theory and practice* (pp. 21–38). Springer.

Fitzgerald, J., Bryans, J., & Payne, R. (2012). A formal model-based approach to engineering systems-of-systems. In L. M. Camarinha-Matos, L. Xu, & H. Afsarmanesh (Eds.), *Collaborative networks in the internet of services* (Vol. 380, pp. 53–62). IFIP Advances in Information and Communication Technology. Springer Berlin Heidelberg.

Fitzgerald, J., Foster, S., Ingram, C., Larsen, P. G., & Woodcock, J. (2013). *Model-based engineering for systems of systems: The compass manifesto* (tech. rep. No. Manifesto Version 1.0). Comprehensive Modelling for Advanced Systems of Systems - COMPASS. Cambrigde, United Kingdom.

Gassara, A., Bouassida, I., & Jmaiel, M. (2017). A tool for modeling sos architectures using bigraphs. In *32nd Symposium on Applied Computing (SAC 2017)* (pp. 1787–1792). Marrakech, Morocco: ACM.

Gassara, A., Rodriguez, I. B., Jmaiel, M., & Drira, K. (2017). A bigraphical multi-scale modeling methodology for system of systems. *Computers & Electrical Engineering*, *58*(Supplement C), 113–125.

Gokhale, A., Balasubramanian, K., Krishna, A. S., Balasubramanian, J., Edwards, G., Deng, G., ... Schmidt, D. C. (2008). Model driven middleware: A new paradigm for developing distributed real-time and embedded systems. *Science of Computer Programming*, *73*(1), 39–58.

Graciano Neto, V. V. (2016). Validating emergent behaviors in systems-of-systems through model transformations. In *ACM Student Research Competition at MODELS* (pp. 1–6). Saint Malo, France: CEUR.

Graciano Neto, V. V. (2018). *A simulation-driven model-based approach for designing software-intensive systems-of-systems architectures* (Doctoral dissertation, University of São Paulo). doi:10.11606/T.55.2018.tde-06072018-110150

Graciano Neto, V. V., Cavalcante, E., Hachem, J. E., & Santos, D. S. (2017). On the interplay of business process modeling and missions in systems-of-information systems. In *IEEE/ACM Joint 5th International Workshop on Software Engineering for Systems-of-Systems and 11th Workshop on Distributed Software Development, Software Ecosystems and Systems-of-Systems (JSOS 2017)* (pp. 72–73). Buenos Aires, Argentina: IEEE.

Graciano Neto, V. V., Garcés, L., Guessi, M., Paes, C., Manzano, W., Oquendo, F., & Nakagawa, E. Y. (2018). ASAS: An approach to support simulation of smart systems. In *51st Hawaii International Conference on System Sciences (HICSS 2018)* (pp. 5777–5786). Big Island, Hawaii, USA: IEEE.

Graciano Neto, V. V., Horita, F. E. A., Cavalcante, E., Rohling, A. J., Hachem, J. E., Santos, D. S., & Nakagawa, E. Y. (2018). A study on goals specification for systems-of-information systems: Design principles and a conceptual model. In *14th Brazilian Symposium on Information Systems (SBSI 2018)* (pp. 1–8). Caxias do Sul, Brazil. doi:10.1145/3229345.3229369

Graciano Neto, V. V., Manzano, W., Rohling, A., Volpato, T., & Nakagawa, E. Y. (2018). Externalizing patterns for simulation in software engineering of systems-of-systems. In *ACM/SIGAPP Symposium On Applied Computing (SAC 2018)* (pp. 1–8). Pau, France: ACM.

Graciano Neto, V. V., Oquendo, F., & Nakagawa, E. Y. (2017). Smart systems-of-information systems: Foundations and an assessment model for research development. In R. Araujo, R. Maciel, & C. Boscarioli (Eds.), *Grand Challenges in Information Systems for the next 10 years* (pp. 1–12). Porto Alegre, Brazil: SBC.

Graciano Neto, V. V., Paes, C. E., Garcés, L., Guessi, M., Oquendo, F., & Nakagawa, E. Y. (2017). Stimuli-SoS: A model-based approach to derive stimuli generators in sim-

ulations of software architectures of systems-of-systems. *Journal of the Brazilian Computer Society*, *23*(1), 13:1–13:22.

Griendling, K. & Mavris, D. N. (2011). Development of a dodaf-based executable architecting approach to analyze system-of-systems alternatives. In *32nd Aerospace Conference (AeroConf 2011)* (pp. 1–15). Big Sky, USA: IEEE.

Guessi, M., Cavalcante, E., & Oliveira, L. B. R. (2015). Characterizing Architecture Description Languages for Software-Intensive Systems-of-Systems. In *3rd International Workshop on Software Engineering for Systems-of-Systems (SESoS) at the 37th International Conference on Software Enginnering (ICSE)* (pp. 12–18). Florence, Italy.

Guessi, M., Graciano Neto, V. V., Bianchi, T., Felizardo, K. R., Oquendo, F., & Nakagawa, E. Y. (2015). A systematic literature review on the description of software architectures for systems of systems. In *30th Symposium on Applied Computing (SAC 2015)* (pp. 1433–1440). Salamanca, Spain: ACM.

Guessi, M., Oquendo, F., & Nakagawa, E. Y. (2016). Checking the architectural feasibility of systems-of-systems using formal descriptions. In *11th System of Systems Engineering Conference (SoSE 2016)* (pp. 1–6). Kongsberg, Norway: IEEE.

Hause, M. (2010a). Model-based system of systems engineering with updm. In *20th Annual International Symposium of the International Council on Systems Engineering (INCOSE 2010)* (Vol. 1, pp. 580–594). Chicago, Illinois, USA: INCOSE.

Hause, M. (2010b). The unified profile for DoDAF/MODAF (UPDM) enabling systems of systems on many levels. In *4th ieee international systems conference (syscon 2010)* (pp. 426–431). Xiamen, China: IEEE.

INPE. (2019). Sistema Integrado de Dados Ambientais. http://sinda.crn.inpe.br/PCD/SITE/novo/site/index.php. Accessed: January 2019. Instituto Nacional de Pesquisas Espaciais.

Iqbal, M. Z., Arcuri, A., & Briand, L. (2015). Environment modeling and simulation for automated testing of soft real-time embedded software. *Software & Systems Modeling*, *14*(1), 483–524. doi:10.1007/s10270-013-0328-6

ISO/IEC/IEEE 42010. (2011). *International Standard for Systems and Software Engineering – Architectural description*.

Issarny, V. & Bennaceur, A. (2013). Composing distributed systems: Overcoming the interoperability challenge. In E. Giachino, R. Hähnle, F. S. de Boer, & M. M. Bonsangue (Eds.), *11th International Symposium on Formal Methods for Components and Objects* (Vol. LNCS 7866, pp. 168–196). Bertinoro, Italy: Springer Berlin Heidelberg.

Klein, J. & van Vliet, H. (2013). A systematic review of system-of-systems architecture research. In *9th international acm sigsoft conference on quality of software architectures (qosa 2013)* (pp. 13–22). Vancouver, Canada: ACM.

Kruchten, P. (1995). Architectural Blueprints - The 4+1 View Model of Software Architecture. *IEEE Software*, *12*(6), 42–50.

Kruchten, P. (2009). Documentation of Software Architecture from a Knowledge Management Perspective – Design Representation. In M. A. Babar, T. Dingsøyr, P. Lago, & H. van Vliet (Eds.), *Software architecture knowledge management theory and practice* (pp. 39–57). Springer.

Lago, P., Malavolta, I., Muccini, H., Pelliccione, P., & Tang, A. (2015). The road ahead for architectural languages. *IEEE Software*, *32*(1), 98–105.

Laudon, K. C. & Laudon, J. P. (2015). *Management information systems: Managing the digital firm* (14th). Upper Saddle River, NJ, USA: Pearson/Prentice Hall.

Maier, M. W. (1998). Architecting principles for systems-of-systems. *Systems Engineering*, *1*(4), 267–284.

Majd, S., Marie-Hélène, A., & Alok, M. (2015). An architectural model for system of information systems. In I. Ciuciu et al. (Eds.), *OTM 2015 Workshops* (Vol. LNCS 9416, pp. 411–420). Rhodes, Greece: Springer.

Manzano, W., Graciano Neto, V. V., & Nakagawa, E. Y. (2019). Dynamic-SoS: An approach for the simulation of system-of-systems dynamic architectures. *The Computer Journal*, 1–23. (in press).

Medvidovic, N. & Taylor, R. N. (2000). A classification and comparison framework for software architecture description languages. *IEEE Transactions on Software Engineering*, *26*(1), 70–93.

Mens, T., Magee, J., & Rumpe, B. (2010). Evolving Software Architecture Descriptions of Critical Systems. *Computer*, *43*, 42–48.

Mittal, S. & Risco Martin, J. (2013). Model-driven systems engineering for netcentric system of systems with DEVS unified process. In *45th Winter simulation Conference (WSC 2013)* (pp. 1140–1151). Washington DC, USA: Society for Modeling and Simulation International.

Nielsen, C. B., Larsen, P. G., Fitzgerald, J., Woodcock, J., & Peleska, J. (2015). Systems of Systems Engineering: Basic Concepts, Model-Based Techniques, and Research Directions. *ACM Computing Surveys*, *48*(2), 18:1–18:41.

Oquendo, F. (2016a). π-Calculus for SoS: A Foundation for Formally Describing Software-intensive Systems-of-Systems. In *11th IEEE System of Systems Engineering Conference (SoSE 2016)* (pp. 1–6). Kongsberg, Norway: IEEE.

Oquendo, F. (2016b). Formally Describing the Software Architecture of Systems-of-Systems with SosADL. In *11th ieee system of systems engineering (SoSE 2016)* (pp. 1–6). Kongsberg, Norway: IEEE.

Oquendo, F. (2016c). Software architecture challenges and emerging research in software-intensive systems-of-systems. In *10th european conference on software architecture (ECSA 2016)* (pp. 3–21). Copenhagen, Denmark: Springer.

Paes, C. E. B., Graciano Neto, V. V., Moreira, T., & Nakagawa, E. Y. (2019). Conceptualization of a system-of-systems in the defense domain: An experience report in the brazilian scenario. *IEEE Systems Journal*, 1–10. doi:10.1109/JSYST.2018.2876836

Pelliccione, P., Kobetski, A., Larsson, T., Aramrattana, M., Aderum, T., S. M., ... Thorsén, A. (2016). Architecting cars as constituents of a system of systems. In *International Colloquium on Software-intensive Systems-of-Systems at 10th European Conference on Software Architecture (SiSoS@ECSA 2016)* (pp. 1–7). Copenhagen, Denmark: ACM. doi:10.1145/3175731.3175733

Rech, A., Pistauer, M., & Steger, C. (2018). Increasing interoperability between heterogeneous smart city applications. In *International conference on internet and distributed computing systems* (pp. 64–74). Springer.

Saleh, M. & Abel, M.-H. (2015). Information systems: Towards a system of information systems. In *7th Conference on Knowledge Discovery, Engineering and Management (IC3K)* (pp. 193–200). Lisbon, Portugal: SciTePress.

Santos, D. S., Oliveira, B. R. N., Duran, A., & Nakagawa, E. Y. (2015). Reporting an experience on the establishment of a quality model for systems-of-systems. In *The 27th international conference on software engineering and knowledge engineering (SEKE 2015)* (pp. 304–309). Pittsburgh, PA, USA: Knowledge Systems institution.

Silva, E., Cavalcante, E., & Batista, T. (2017). Refining missions to architectures in software-intensive systems-of-systems. In *IEEE/ACM Joint 5th International Workshop on Software Engineering for Systems-of-Systems and 11th Workshop on Distributed Software Development, Software Ecosystems and Systems-of-Systems (JSOS 2017)* (pp. 2–8). Buenos Aires, Argentina: IEEE.

Tomicic-Pupek, K., Dobrovic, Z., & Furjan, M. T. (2012). Strategies for information systems integration. In *34th International Conference on Information Technology Interfaces (ITI 2012)* (pp. 311–316). USA: IEEE.

Wachholder, D. & Stary, C. (2015). Enabling emergent behavior in systems-of-systems through bigraph-based modeling. In *10th International Conference on Systems of Systems Engineering (SoSE 2015)* (pp. 334–339). San Antonio, TX, USA: IEEE.

Wiederhold, G. (1992). Mediators in the architecture of future information systems. *Computer*, *25*(3), 38–49.

Woodcock, J., Cavalcanti, A., Fitzgerald, J., Larsen, P., Miyazawa, A., & Perry, S. (2012). Features of CML: A formal modelling language for Systems of Systems. In *7th International Conference on System of Systems Engineering (SoSE 2012)* (pp. 1–6). Genova, Italy.

Yamaguti, W., Orlando, V., & Pereira, S. (2009). Sistema brasileiro de coleta de dados ambientais: Status e planos futuros. *Simpósio Brasileiro de Sensoriamento Remoto*, *14*, 1633–1640.

Zeigler, B. P., Kim, T. G., & Praehofer, H. (2000). *Theory of modeling and simulation* (2nd). Orlando, FL, USA: Academic Press, Inc.

Authors Biography

Milena Guessi is a post-doctoral research fellow at the University of São Paulo (ICMC-USP), São Carlos, Brazil. She obtained her B.Sc. in Computer Science and her M.Sc. in Software Engineering from USP. She received her Ph.D. degree in Computer Science from USP and the University of South Brittany (UBS), France, in 2017. Her main research interests include software architectures, architecture descriptions, and systems-of-systems (SoS). She has organized local and international events and has served as a program committee member of conferences and as a referee of journals. She is a member of ACM and SBC.

Valdemar Vicente Graciano Neto is an assistant professor with the Informatics Institute at the Federal University of Goiás, Goiânia, Brazil. He received his Ph.D. degree in computer science and computational mathematics from the Institute of Mathematical Sciences and

Computation, University of São Paulo (USP), Brazil and the Docteur degree in sciences and information technology from Université Bretagne-Sud (UBS), Lorient, France, in 2018. He is also the President of the Special Committee on Information Systems of the Brazilian Computer Society (SBC) in the 2018–2019 period. Dr. Graciano Neto is a member of the SBC, the Society for Modeling and Simulation International (SCS) and of ACM.

Elisa Yumi Nakagawa is an associate professor in the Department of Computer Systems at the University of São Paulo - USP, Brazil. She conducted her post-doctoral research at Fraunhofer IESE, Germany, in 2011–2012 and at the University of South Brittany, France, in 2014-2015. She received her Ph.D. degree from USP in 2006. She coordinates international/national research projects in software architecture, reference architectures, and systems-f-systems, and also supervises PhD/-Master's students and post-doctoral researchers. She has organized international/national conferences and has served as a program committee member at many conferences and as a reviewer of various journals. She is a member of IEEE and SBC.