Graciano Neto *et al. Journal of the Brazilian Computer Society* (2017) 23:13
DOI 10.1186/s13173-017-0062-y

Journal of the
Brazilian Computer Society

RESEARCH Open Access

Stimuli-SoS: a model-based approach to derive stimuli generators for simulations of systems-of-systems software architectures

Valdemar Vicente Graciano Neto[1,2,3*], Carlos Eduardo Barros Paes[4], Lina Garcés[1,2], Milena Guessi[1,2], Wallace Manzano[1], Flavio Oquendo[2] and Elisa Yumi Nakagawa[1]

Abstract

Background: Systems-of-systems (SoS) are alliances of independent and interoperable software-intensive systems. SoS often support critical domains, being required to exhibit a reliable operation, specially because people's safety relies on their services. In this direction, simulations enable the validation of different operational scenarios in a controlled environment, allowing a benchmarking of its response as well as revealing possible breaches that could lead to failures. However, simulations are traditionally manual, demanding a high level of human intervention, being costly and error-prone. A stimuli generator could aid in by continuously providing data to trigger a SoS simulation and maintaining its operation.

Methods: We established a model-based approach termed *Stimuli-SoS* to support the creation of stimuli generators to be used in SoS simulations. Stimuli-SoS uses software architecture descriptions for automating the creation of such generators. Specifically, this approach transforms SoSADL, a formal architectural description language for SoS, into dynamic models expressed in DEVS, a simulation formalism. We carried out a case study in which *Stimuli-SoS* was used to automatically produce stimuli generators for a simulation of a flood monitoring SoS.

Results: We run simulations of a SoS architectural configuration with 69 constituent systems, i.e., 42 sensors, 9 crowdsourcing systems, and 18 drones. Stimuli generators were automatically generated for each type of constituent. These stimuli generators were capable of receiving the input data from the database and generating the expected stimuli for the constituents, allowing to simulate constituent systems interoperations into the flood monitoring SoS. Using Stimuli-SoS, we simulated 38 days of flood monitoring in little more than 6 h. Stimuli generators correctly forwarded data to the simulation, which was able to reproduce 29 flood alerts triggered by the SoS during a flooding event. In particular, Stimuli-SoS is almost 65 times more productive than a manual approach to producing data for the same type of simulation.

Conclusions: Our approach succeeded in automatically deriving a functional stimuli generator that can reproduce environmental conditions for simulating a SoS. In particular, we presented new contributions regarding productivity and automation for the use of a model-based approach in SoS engineering.

Keywords: Simulation, Software architecture, Systems-of-systems, Automatic generation, Model transformation

*Correspondence: valdemarneto@usp.br
[1]University of São Paulo, Av. Trabalhador Sancarlense, 400, 13566-590 São Carlos, Brazil
[2]University of South Brittany, Rue André Lwoff, 56000 Vannes, France
Full list of author information is available at the end of the article

Introduction

Systems-of-systems (SoS)[1] are a set of interoperable systems called constituents joined together to accomplish complex missions [1–4]. SoS often support missions in critical domains, such as smart cities, traffic control, and emergency and crisis response management [5–8]. Substantial investments have been made to support SoS engineering. For instance, Saudi Arabia has invested 70 billion dollars in smarter cities, and South Africa has recently started a 7.4 billion dollars project on smart cities [9]. Due to the critical nature of domains that SoS intend to support, SoS exhibit a noteworthy risk of causing damage, financial losses, and threats to human life. Hence, they must be constructed to be trustworthy, i.e., their operation must be reliable so that people can rely on their services to accomplish their own missions correctly, without failing nor causing accidents, working as expected, and keeping their operation in progress [10–12].

Simulations can contribute to guarantee SoS trustworthiness. They consist of a recurrent approach in SoS Engineering to anticipate failures early in SoS life cycle. Simulations externalize how the whole SoS behaves at runtime [13–16]. To be reliable, a simulation must faithfully reproduce the conditions under which a SoS operates. These conditions must involve SoS surrounding environment (such as rain and temperature) and constituent operational conditions (such as battery level and GPS location) [15, 17]. A manual approach can fail to reproduce the real frequency of such stimuli, since an expert would have to simultaneously inform inputs for all constituents at runtime until the end of the simulation. Moreover, a manual approach to generate inputs for such simulation can be costly. For example, to reproduce SoS dynamics, for each unit of time, each constituent in the simulation must be fed. A stimulus is often delivered to a constituent system through a user interface interaction. For each stimulus, one user interaction is needed. Considering a SoS formed by six constituents, if each one of them requires one stimulus by unit of time, after only 100 units of time, 600 interactions (such as clicks) need to be performed. Thus, the effort needed to feed a simulation with a greater number of constituents or for a longer period of time is extremely high, making this approach unfeasible to simulate real SoS.

In this scenario, stimuli generators can support SoS simulation. They consist of a virtual simulation entity responsible for playing the role of the environment, delivering input to a SoS [18].

However, manually coding of such stimuli generator is equivalently not feasible. Stimuli generators are domain-dependent and totally adherent to the environment modeling, which is itself challenging for SoS development [19–21]. For example, if we want to simulate a reactive system (such as a temperature sensor), it is important to predict a subset of stimuli that it can receive in order to establish how it will react to them. This entity should encompass details such as the scale in which it will work (celsius, fahrenheit, kelvin, or another scale), a range of acceptable values (from −50 to 60 °C, for example), the description of the data as a data structure (with value and type), instances that could be received, and frequency in which it must be delivered. Additionally, its development is costly, as it requires writing additional simulation code, often in a lower abstraction level, such as state machines, ports, inputs, and outputs details. Aiming to reduce costs associated to the engineering of a stimuli generator, we can explore the possibility of automating its creation, hence supporting (i) the prediction of the surrounding environment dynamics and (ii) an anticipation of possible events and natural phenomena that could hamper SoS correct operation.

In context, it is noteworthy to pose the following research question: *How is it possible to automatically obtain a functional stimuli generator that reproduces environmental conditions to the simulation of a SoS*? To answer this question, in this article we present a model-based derivation approach for automatically producing stimuli generators to feed a SoS simulation at runtime. In this approach, architectural descriptions play the role of input model as they inherently store information about expected inputs and outputs of the SoS, supporting environmental modeling. We evaluate our approach with regard to its *correctness/reliability* in automatically producing stimuli generators for the simulation of a real SoS that monitors flash floods risk in a river that crosses urban areas. Results of this study reveal that our approach is reliable and capable of deriving stimuli generators that conform with the expected inputs that must be received by simulated constituents and that effectively triggers the SoS simulation.

This paper is structured as follows: "Background" section briefly introduces the foundations for understanding our approach, "Presentation of Stimuli-SoS" section presents our approach Stimuli-SoS, "Evaluation" section details the case study, "Discussion" section discusses our results and related work, and "Final remarks and forthcoming steps" section has our final remarks, discussing potential future works.

Background

Modelling and simulation (M&S) are vital elements within processes for analysis and design of SoS. M&S enable visualization of SoS dynamics [17, 22–25]. Several application domains adopt M&S [17]. Simulations correspond to an imitation of the operation of a real-world process or system over time and involve generation of artificial stimuli and the observation of its outcome to draw inferences about the operation of real systems that they

represent [15, 17, 26]. As such, M&S promote (i) a visual and dynamic viewpoint for SoS software architectures, reproducing stimuli the system can receive from a real environment, (ii) prediction of errors, diagnosing them and enabling corrections, and (iii) observation of expected and unexpected emergent behaviors of an SoS [27, 28].

SoS must be analyzed under a multitude of viewpoints, and these viewpoints can be distinguished into two families: static approaches, focusing on systems properties, and dynamic approaches, focusing on their behavior [23]. As SoS exhibit emergent behaviors, dynamic approaches are especially interesting. An emergent behavior can be classified under two perspectives [14]: *Intention* and *Type*. An emergent behavior can be *Predicted* or *Unpredicted* [29]. Predicted emergent behavior consists of behaviors intentionally designed to emerge at runtime, whilst unpredicted emergent behavior corresponds to that one that emerges as a co-lateral effect of specific conditions or runtime configurations, with the potential to cause losses to the SoS operation. Considering the type, four categories exist [14]: *Simple*, *predicted*, *strong*, and *spooky*. *Simple* emergent behaviors are emergent properties readily predicted by simplified models of the SoS. They are produced in lower complexity through models that abstract the SoS (only intentional predicted behaviors emerge since the model is overly simple). *Predicted* emergent behaviors are those readily and consistently reproducible in simulations of the system, but not in static models. They are partially predicted in advance (desired behaviors are predicted, but undesired can also appear). *Strong* emergent behaviors are consistent with SoS known properties, but are not reproducible in any model of the system. Direct simulations may reproduce the behavior, but inconsistently, and simulations do not predict where the property will occur (desired behaviors exist, but unpredicted behaviors are the majority). Finally, *spooky* emergence is inconsistent with known properties of the SoS, not reproducible or subject to simulation (a natural emergence, such as life itself, not predicted).

Baldwin et al. summarize current techniques found in the literature to simulate SoS [24]. Event-based modeling is the most prominent approach, as researchers can program different states a system undergoes to comprehend the behavior of the SoS as a whole [24]. In particular, DEVS is the most popular event-based simulation formalism [30]. It represents SoS, providing the required dynamic view of SoS. However, a straightforward generation of DEVS code does not guarantee that the simulation is executable. This happens because the SoS operation is deeply related to the stimuli received from the environment that triggers the simulation execution. Hence, it is necessary to elaborate a specific entity in the simulation model that is responsible for delivering expected stimuli that drive the operation of the SoS: the stimuli generator.

Regardless of the approach adopted to simulate SoS, simulations often depend on some internal structure that imitates the surrounding environment of an SoS, delivering stimuli that are assumed to be received by the SoS to trigger its operation [31]. The environment comprises the SoS surroundings, such as temperature, wind, water level, and noise, and/or conditions in which a system operates, such as battery level and geographic position [4]. Environment is local to each system. By the nature of SoS, environments are only partially known at design time [8].

There are two alternatives to deliver stimuli to a simulation [18, 32–35]. The first one is adding a portion of code to the body of each constituent in the simulation, randomly producing data [36]. However, this approach brakes the separation of concerns principle, decreasing maintainability, as this code will be tangled to the constituent operational code. The second alternative is to materialize all stimuli into a single artificial entity known as *stimuli generator*. This structure becomes part of the simulated SoS, continuously delivering stimuli to SoS. Hence, stimuli generators imitates the SoS surrounding environment, automating the stimuli input [33–35, 37–39].

Developing stimuli generators require a careful investigation of SoS requirements and architecture specification to elicit which stimuli should be provided. Such tasks can bring additional cost to the SoS development and might be error-prone when a manual approach is used to transform architectural elements in software code. Moreover, stimuli generator can be used as an interface between the simulator actually employed and other industrial simulators used to imitate real environments, such as flight simulators, or a river simulation for flood monitoring SoS. This association between two types of simulator is known as *co-simulation* [40, 41]. This approach is broadly adopted by the industry to a large-scale test. Meanwhile, despite the potential of stimuli generators to support co-simulation approaches in SoS development, such approaches for automatically creating this stimuli generator for simulation of SoS have not been widely investigated.

Model-based engineering (MBE) techniques have been investigated in the context of SoS [16, 22, 25, 42, 43]. They represent a software engineering approach in which models are the main basis, spanning all activities that make up the software development process [6, 44]. MBE has been supported by a broad set of tools that are available to achieve a proper level of automation using transformation tools, such as Xtend [45] and Acceleo [46]. Model transformations are the heart of MBE [44]. Model transformations are a well-accepted approach that aids software engineers in establishing correspondences between models [47]. It consists of a program, often written in a declarative manner, that transforms an input model in an output model [44]. MBE can be exploited to generate stimuli generators. Next section details our approach.

Presentation of Stimuli-SoS

Stimuli-SoS is a model-based approach established to automatically derive stimuli generators for SoS simulations. For each distinct type of constituent in a SoS, a dedicated stimuli generator is created. Architectural models often bring some sort of environment description [48]. To establish the basis of Stimuli-SoS, we decided to use a SoS architectural model to derive stimuli generators for a SoS simulation via a model transformation. We surveyed the literature to find a formalism that could properly support SoS software architecture modeling and simulation [49]. For this purpose, such formalism should meet the following language requirements, supporting:

1. SoS simulation,
2. Specification of dynamic architectures,
3. Multiple constituents modeling,
4. Constituents interoperability modeling, and
5. SoS software architecture specificities and precision, including environment modeling

Table 1 summarizes the comparison between potential formalisms to support SoS software architecture representation, simulation, and stimuli generator derivation.

We decided to search for a software architecture notation, an ADL or modeling notation, that could support all the concepts necessary to represent SoS and that could be simulated. The following modeling languages have been identified as the key ones used for SoS architecture description: Darwin (semi-formal) [50], Wright (formal) [51], π-ADL (formal) [52], UML[2] (semi-formal), SysML[3] (semi-formal), and SoSADL [8] [49].

Wright and Darwin were not designed to model SoS architectures. The aforementioned requirements are not covered as these ADL were created to model monolithic systems. SysML was the backbone of two European projects (COMPASS[4] and DANSE[5]) for which they developed extensions for SoSs. DANSE did not develop an ADL, but used SysML for semi-formally describe executable architectures that are then tested against contracts. However, SysML is a UML Profile, and not necessarily an ADL. Moreover, the adoption of SysML to model SoS would require multiple models, each one being simulated individually, and the simulations being interoperated, what is costly. Then, SysML does match our approach requirements, and despite being adopted for software architecture modeling, is not strictly an ADL. UML shares the same drawbacks.

Table 1 Comparison between formalisms for SoS simulation and software architecture specification considering the aforementioned language requirements

Approach	1	2	3	4	5
Simulink/MATLAB	Yes	No	Yes	Yes	No
SysML	Yes	No	Yes	Yes	No
UML/Executable UML	Yes	No	Yes	No	No
DEVS [15]	Yes	Yes	Yes	Yes	No
CML	Yes	No	Yes	Yes	No
Darwin	No	Yes	No	No	No
Wright	No	Yes	No	No	No
π-ADL	No	Yes	Yes	Yes	No
SoSADL	No	Yes	Yes	Yes	Yes

COMPASS developed a formal approach, in contrast to DANSE that extended a semi-formal one. In COMPASS, CML was specifically designed for SoS modeling and analysis. However, CML is not an ADL. It is a contract-based formal specification language to complement SysML: SysML is used to model the constituent systems and interfaces among them in a SoS, and CML is used to enrich these specifications with contracts. A CML model is defined as a collection of process definitions (based on CSP/Circus), which encapsulate state and operations written in VDM as well as interactions via synchronous communications. CML is a low-level formal language, of which a key drawback is that SysML models when mapped to CML results in huge unintelligible descriptions (it was one of the lessons learned from COMPASS) [8].

Finally, π-ADL is a formal language to model distributed architectures. However, despite π-ADL providing architectural description models for concurrent and communication processes, it does not provide straightforward abstractions for some SoS' particular concepts, such as mediator, coalition, and environment modeling, and it does not support modeling of abstract architectures and simulations. SoSADL is more expressive than π-ADL for the description of SoS software architectures, with additional elements, such as gates, duties, guarantees, properties, and mediators. In SoSADL, architectural descriptions are intentional and abstract, whereas in π-ADL, such descriptions are declarative and concrete. In addition, from a formal point of view, SoSADL includes other formalisms besides the π-calculus, which is the only one that π-ADL possesses.

Hence, only SoSADL matched the majority of requirements we raised. SoSADL is a language formally founded on π-calculus for SoS, a novel process calculus extended from original π-calculus, conceived for enabling the formal architecture description of software-intensive SoS. It can be considered correct by construction, as the formal semantics of such calculus is defined by means of a formal transition system, expressed as labeled transition rules, which are formulated as proof rules [53]. In short, SoSADL describes SoS, which can be expressed

as a combination of architecture, systems, and mediators declarations[6]. Each architecture declaration is expressed in terms of its intrinsic behavior, data types, and gates, i.e., abstractions that enable the establishment of connections. A connection is established to receive stimulus from or act on the environment, or to simply communicate with other constituents. Furthermore, a connection can be used to receive, send, or do both actions. Data types can have inherent functions, and functions can have associated expressions. Mediators and systems as well as the SoS architecture itself also have gates, data types, and behaviors. Systems play the role of constituents in an Architecture Behavior Declaration, and systems are mediated by mediators. SoSADL supports representation of emergent behavior by means of a coalition, i.e., a temporary alliance that allows constituents to perform a combined action. These emergent behaviors are specified as part of the coalition behavior, documenting how constituents should interact to accomplish a given set of missions[7].

However, as SoSADL is not executable yet, we needed to combine SoSADL with the simulation formalism that matched the majority of our requirements and complemented the characteristics not covered by SoSADL. SosADL holds the environment description, whilst a simulation formalism run the stimuli generators created.

Simulation formalisms (or notations subject to simulation) such as Simulink/MATLAB[8], Executable UML [54], or SysML[9] do not support any one of the aforementioned requirements. Even if such formalisms could represent SoS constituents by means of multiple models, each one representing one individual constituent; hence, simulating them would require interoperability between individual simulations (known as distributed simulations), which demands further adaptations and costly implementations. DEVS, in turn, was developed specially to represent SoS architectures. Hence, we chose DEVS as the formalism to simulate our SoS software architectures.

However, even DEVS lacks important characteristics for expressing SoS software architectures, such as (i) the language only deals with the notion of ports; there is no notion of connections and gates separately that is a remarkable paradigm of software architectures (Components, Connections, and Values/Constraints define a software architecture [48, 55]) used by SoS modeling [56], (ii) in DEVS, every major entity of an architecture is represented as the same type of model (called atomic model). As the single abstraction available, an atomic model prevents a complete characterization of the software architecture with the diversity and typical heterogeneity of constituents that form a SoS, (iii) even though it supports environment modeling, its inherent syntax does not have any specific mechanism for representing the surrounding environment, which is an important aspect of any software architecture, including software architectures of SoS [48], (iv) it does not offer a mechanism to automatically create stimuli generators, and finally, (v) since SoS architectures are dynamic, i.e., their constituents are not necessarily known at design time and they can join or leave the SoS at runtime, it lacks the notion of abstract architectures, i.e., a description of constituents and their *potential connections* with other constituents, and how they could be adapted at runtime. Even though DEVS supports dynamic reconfiguration, i.e., the modeling and simulation of architectures of SoS that adapt themselves at runtime, the language still lacks support for abstract architectures, thus requiring all constituents that take part in the simulation to be known. Therefore, it is not allowed for a new constituent, i.e., a constituent that has not been predicted at design time, to join the coalition at runtime.

We thus harmonize both formalisms SoSADL and DEVS, by means of a model transformation (SoSADL2DEVS), to automatically derive the stimuli generators from the SoS software architectural descriptions documented in SoSADL. Figure 1 illustrates how the transformation is carried out. A SoSADL model is

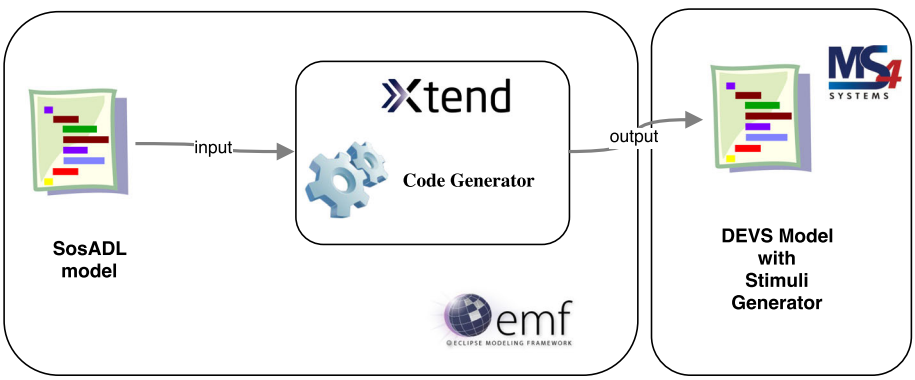

Fig. 1 SoSADL2DEVS transformation [77]

used as input to an Xtend script that materializes a model transformer. In our approach, there is an Xtend script that materializes the Code Generator. A functional code written in DEVS is generated as output, establishing a SoSADL2DEVS transformation. Considering a broad view of the transformation, the concepts of system and mediator in SoSADL are transformed into atomic models in DEVS. Behaviors are materialized into labeled state diagrams in DEVS that configures the constituents operations. Architecture, coalition, and SoS become coupled models. Connections and gates become ports, and data types and functions are mapped, respectively, in data types and functions in DEVS.

A systematic approach to derive stimuli generators

We established Stimuli-SoS as a systematic approach based on well-defined activities. The systematic approach involves a reference workflow to derive stimuli generators. Figure 2 shows the proposed workflow which is represented through an UML activity diagram using SPEM[10] stereotypes. Each activity is developed by a SoS architect. The result of the execution is the generation of work products. Our approach consists of the following activities:

1. *Specification of SoS software architecture*: In the first activity, an architectural description of the SoS software architecture is specified using SoSADL. The work products delivered are used as input for the next activity. Environment modeling is a sub-activity performed in this step
2. *Automatic derivation of stimuli generators*: This activity comprises running the model transformation, receiving SoSADL work products as inputs, and delivering DEVS files as outputs, including the stimuli generators

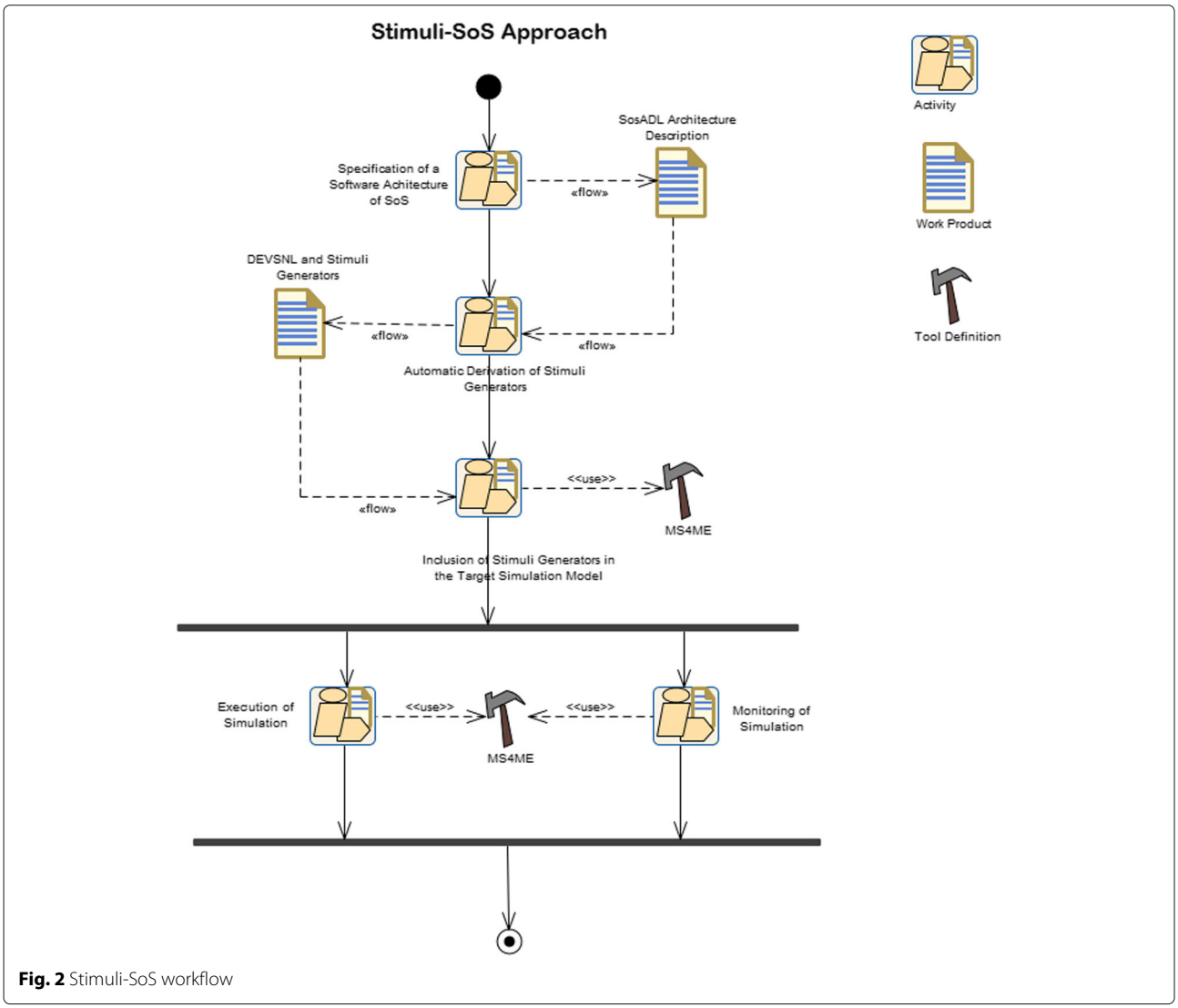

Fig. 2 Stimuli-SoS workflow

3. *Inclusion of stimuli generators in the target simulation model*: After delivering the DEVS files, they must be included in a project that is deployed in MS4ME tool to support the launching and execution of the simulation; and
4. *Execution and monitoring of simulation*: This activity uses the stimuli generator and collects data from the simulation to observe emergent behaviors, to perform statistical analysis and to collect evidence for validation and verification of properties of the SoS software architecture.

Model transformation

All SoSADL elements must be mapped to DEVS to create a functional simulation. In SoSADL, there is a special type of connection called **environment**, which abstracts interaction of a SoS with the surrounding environment, emitting outputs to the environment, or receiving stimuli from it, e.g., when the system is a sensor. However, there are no straightforward elements in DEVS to automatically produce environment stimuli. Thus, it is necessary to create a stimuli generator that delivers the expected inputs the constituents wait to perform transitions and to start their execution.

Listing 1 shows an excerpt of a code in SoSADL that depicts part of the specification of one constituent: in this case, a sensor Additional file 1. Some parts are hidden because they do not influence in the derivation of stimuli generation. It is possible to see, for example, that the gate `energy` offers two environment connections (lines 12 and 13): one to receive a `threshold` (a limit of energy that is considered enough to keep the sensor in operation), and `power`, which is used to receive the level of battery available. A connection in SoSADL has a name and a data type that can be transferred through that communication channel. Then, when a connection is specified with the environment modifier, it actually models what is expected to be received from the environment and the data type expected. Each type of constituent requires a different stimuli generator. Then, such architectural model is used as an input for a model transformation that collects the set of environment connections, extracting the data type, and creating one respective output state transition for each one of them. These state transitions are assembled in sequence to form an entire state diagram that will drive the stimuli generator operation. Then, each state transition will deliver one of the expected data to the correspondent constituent whose architectural specification model was used to create that stimuli generator. Each stimuli generator is associated to a data flow that receives data from a textual file. That file holds the data that feeds the constituent. Then, these data are read from the text file and sent to the constituent, triggering the simulation to run. This happens in a periodic and constant rhythm so as to keep the simulation running.

Listing 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
11     gate energy is {
12       environment connection threshold is in { Energy }
13       environment connection power is in { Energy }
14     }
15   }
16 }
```

The following steps are followed by the transformation chain that produces stimuli generators:

1. All connections of all the constituents are mapped into a specification format and saved in a text file
2. Connections are read from the text file and analyzed. Such connections are parsed from the architectural description of the SoS to be in the following format: `measurement::sense;RawData-true`. This first part is the name of the gate in which the connection has been specified. The second part is the name of the connection. The third part represents the data type that can be transferred across that communication channel. The last part of each connections descriptions is a boolean: it has a `true` value if the connection is of the type `environment` and `false` if it is not. The transformation algorithm searches for environment connections. Each connection specified as an `environment` connection produces one transition in the specification of the state diagram in the resulting stimuli generator. Hence, the stimuli generator consists of a special type of system (in the context of the simulation) that has a continuous behavior (a behavior materialized as a loop) to emit stimulus by output state transitions, starting and keeping the SoS in operation.

Listings available externally[11] show and bring explanations about the Xtend code that materializes the model transformation. We evaluate our approach as follows.

Evaluation

To investigate the reliability of Stimuli-SoS approach, we performed a case study, which is an exploratory type of empirical method for investigating a phenomena in its natural environment using data gathered from

few entities (people, organizations, and sensors) [57]. It refers to a contemporary phenomenon observed in its real-world context, and it often adopt multiple sources of evidence.

We evaluate our approach using a Flood Monitoring and Emergency Response SoS. Such SoS includes wireless river sensors, telecommunication gateways, unmanned aerial vehicles (UAVs), Vehicular Ad Hoc Networks (VANETs), Meteorological Centers, Fire and Rescue Services, Hospital Centers, Police Departments, Short Message Service Centers and Social Networks, as described in [8]. This SoS involves the National Center for Natural Disaster Monitoring, which monitors 1000 cities, with 4700 sensors, including 300 hydrological sensors, and 4400 rain gauges. There are 5000 mobile ground stations to monitor land slopes, and data from other agencies are also used, such as the National Water Agency (ANA[12]), Mineral Resources Research Company (CPRM[13]) sensors, and Aeronautical radars, according to the SENDAI International Protocol for Natural Disaster Risk Reduction (2015–2030), such as droughts, landslides and floods, in the case of the Brazilian reality. We will use a subset of this Flood Monitoring and Emergency Response SoS, which is itself an SoS, i.e., the Urban River Monitoring System, henceforth, FMSoS. Next section details our application scenario.

Scenario description

We specified a FMSoS architecture with 42 sensors, 9 crowdsourcing systems, and 18 drones, following the model shown in Fig. 3. Each drone has its own base

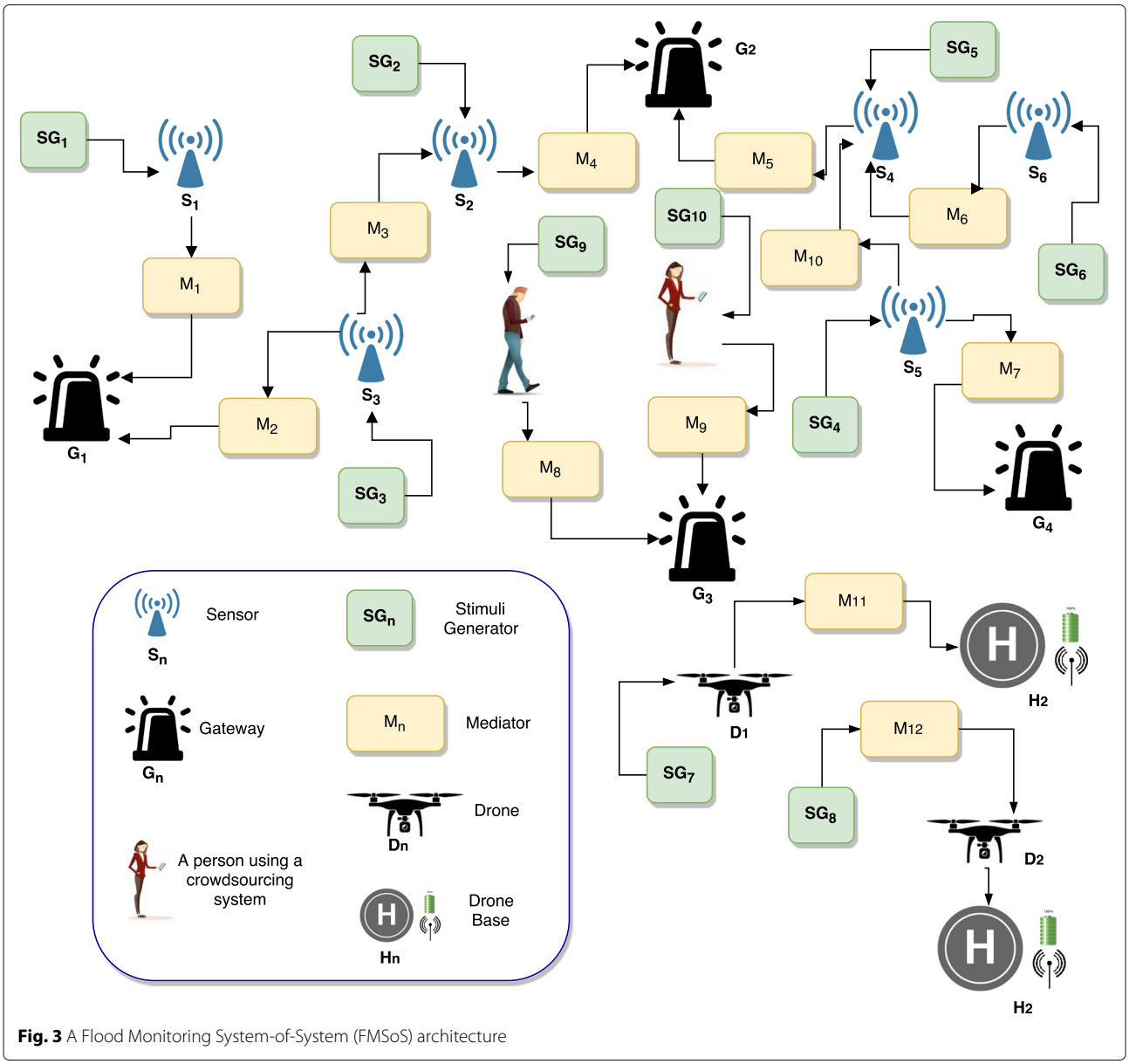

Fig. 3 A Flood Monitoring System-of-System (FMSoS) architecture

(18 drone bases) and transmits the information collected through its own 3G gateway (a gateway that will be in the vicinity). Eighteen gateways are spread along the river boards. Mediators were produced as much as necessary to mediate these constituents, and 20 gateways were also used to receive these transmissions.

FMSoS monitors occurrences of floods in an urban area. Rivers cross the city and, when the rains are intense, floods frequently occur, causing losses, damage, and imminent danger for the population. FMSoS is composed by five different types of constituents:

1. *Smart sensors*, which are fixed embedded systems monitoring flood occurrences in urban areas, located on river edges
2. *Gateways*, which gather data from constituents and share them with other systems
3. *Crowdsourcing systems*, which are mobile applications used by citizens for real-time communication of water level rising
4. *Drones*, which are UAVs also concerned to complement sensors observations by monitoring the river water level while they fly over it, sending pictures if some change in the water level occurs; and
5. *Drone bases*, which are fixed basis from where drones departure, and for where they come back to recharge battery, and transmit their data

Our FMSoS is concerned with one specific mission: *emitting flood alerts to public authorities that can draw strategies to protect the population*. It consists of a collaborative SoS, with no a central authority that orchestrates the constituents functionalities to accomplish missions. Data are gathered in gateways, analyzed according to flood risk, and a status (alert or no alert) is transmitted to public authorities. Figure 4 gives an illustration of FMSoS deployed in a river[14]. Sensors are spread on the river's edges with a regular distance among them, and mediators exist between every pair of sensors in a pre-established distance between them. Data collected by sensors are transmitted until reaching the gateway. Besides, drones fly on the river and return to their bases to recharge and eventually communicate with gateways to alert about a flood threat. In parallel, people that walk close to the river can also contribute by communicating that water level is increasing if they perceive this happening. In case of flood, gateways emit alarms for public authorities. Authorities cross data coming from all the constituents to draw a conclusion of an imminent flood, taking decisions to protect the population.

FMSoS exhibits the following characteristics [1, 8][15]:

- *Operational independence of the constituents*: Each constituent (sensor, crowdsourcing system, or drone) operates in a way that is independent of other constituents, as they belong to different city councils and have different missions in the region of São Carlos.
- *Managerial independence of constituents*: A diversity of stakeholders and enterprises might independently own, deliver, and manage different constituents that compose FMSoS. Moreover, each constituent has its own management strategy for transmission vs. energy consumption and will act under the authority of the different city councils.
- *Distribution*: All the constituents interoperate through a communication network.
- *Evolutionary development*: SoS evolves as a consequence of changes in the configuration or functionality of constituents.
- *Emergent behavior*: One unique constituent could not deliver a flood alert by itself. For instance, if only one sensor, or crowdsourcing system or drone performs its activities in an urban area, it could not notify a flood on time, being not effective. It might emit a false alert, since the flood could be limited to another place. Hence, the flood alert is a result of the interoperability among a diversity of constituents working in cooperation, which spread along the riverbank.

For each one of the constituent types, a specific type of stimuli generator was automatically produced using our model transformation approach. For each constituent type, connections were specified in SoSADL with the `environment` modifier to support the automatic derivation of stimuli generators. Mediators do not need a stimuli generator as they receive stimuli from other constituents, and they do not have environment connections. We discuss the rationale behind each one of the environment connections for each type of constituent, as follows:

- *Smart sensors*: battery (power level), coordinate (GPS location), water level, and power threshold.
  Rationale: they receive battery level and power threshold, and a coordinate to start their work. After that, the stimuli generator will deliver water level to them, imitating the data obtained by sensors to verify if the SoS will detect possible floods.
- *Gateways*: battery (power level), coordinate (GPS location), and power threshold.
  Rationale: The gateway (materialized by an industrial computer linked to the internet) provides the base station for collecting these measures and processing them, possibly warning the risk of imminent flood [8]. Data are transmitted from other constituents and gathered in gateways. Hence, only power level, coordinate, and power threshold are necessary.
- *Crowdsourcing systems*: battery (power level), coordinate (GPS location), visual perception, and power threshold.

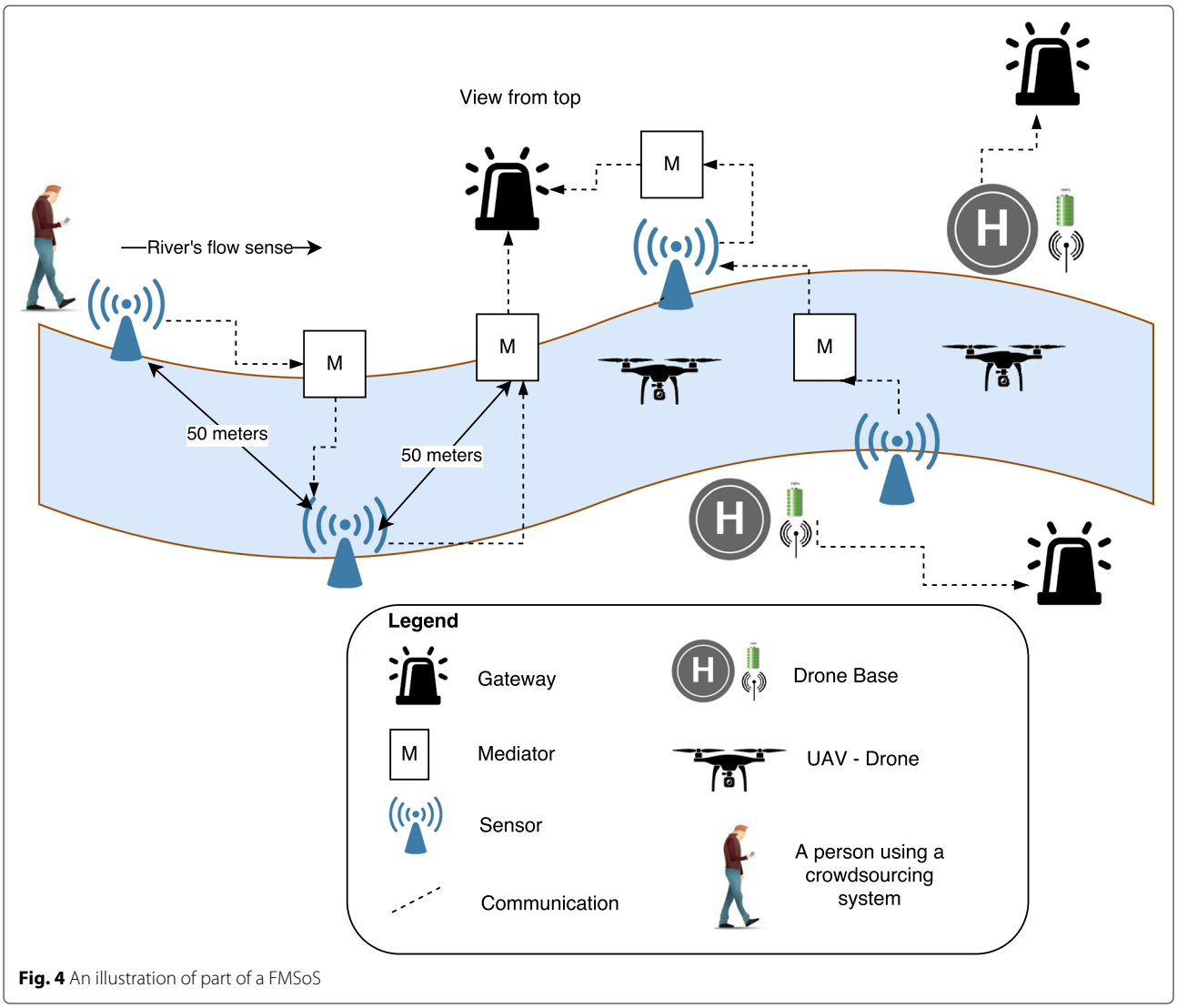

Fig. 4 An illustration of part of a FMSoS

Rationale: crowdsourcing systems are apps that enable population to communicate a possible flood threat by interacting with mobile. It is possible to communicate the risk level and to send pictures to show the situation. These systems do not interact with environment, but with humans. Hence, operational aspects are documented as environment issues (power level, coordinate, and power threshold), and a specified behavior enables citizen to send information according to a pre-defined danger scale and pictures that endorse their perception [58]. However, this perception also represents the environment. Hence, we defined in SoSADL that the danger level is a pre-defined value (between 1 and 6, 1 being no risk, and 6 being flood effectively occurring) that can be classified by the human user according to what he/she sees. Figure 5 shows a real picture of a human dummy painted in a river wall in front of USP. People use it as a reference to classify the flood risks according to the aforementioned levels. In turn, Fig. 6 shows how the numbers and the co-related water level appear in the mobile app so that a person can classify the risk. Looking at the painting available in Fig. 5, it can classify the risk according to the scale available in Fig. 6, and send to gateways. Then, an environment connection called `perception` was defined in SoSADL specification, enabling that these pre-defined data can be sent according to what the user selects. Then, it is still possible to automatically create a stimuli generator that delivers these data.

- *Drones*: battery (power level), coordinate (GPS location), water depth, and power threshold, image, and distance flown.
  Rationale: Most professional radio control systems reach 2 km of radius extension. A drone has an average autonomy range of 10 min. After that, it is

Fig. 5 A real picture of a human dummy used to classify flood risk

required to come back and recharge its battery. Its average speed is 16 m per second. Hence, it can fly 5 min to go and return in the next 5 min to recharge. Then, he can fly a route of 2400 m one way, and 2400 m back. As the Monjolinho River, where we are applying the case study, has an extension of 43 km, it will take 18 drones to individually cover 2.4 km each. We will call the drone connection as `water depth`, since it measures the height of water differently from the sensor. In addition, a connection called `image` will be responsible for taking a photo of the place that has altered water height and send via 3G to responsible authorities, initiating the alert. Photos are taken only when the water depth exceeds a given threshold. For measurement purposes, the `flown distance` will also be delivered constantly to the drone, within the limit of 2.4 km. The `GPS position` is also delivered constantly by the stimulus generator, changing its values over time, to simulate the autonomous movement.

- *Drone basis*: battery (power level), coordinate (GPS location), and power threshold.
  Rationale: This is the radio control basis, for where drones come back to recharge battery. Only its own battery level, coordinate, and power threshold are necessary to model its environment of interest.

Case study protocol

The case study was conducted according to the following steps [57]: (i) case study design (preparation and planning for data collection), (ii) execution (collection of evidence), (iii) analysis of collected data, and (iv) reporting.

Scenario: Our case study consists of a Flood Monitoring SoS (FMSoS) concerned to monitor a river that crosses an urban area, aiming to detect potential flash floods, i.e., floods that can occur quickly with huge damage and risk for population. It consists of the description of part of a SoS already in operation in São Carlos, Brazil, monitoring the Monjolinho river that crosses the urban area and that causes recurrent flash floods, causing damage and losses.

Goal: The goal is to evaluate with regard to its correctness if stimuli generators automatically produced are able to trigger and feed a simulation until the end of its execution.

Rationale: Our approach was designed to support simulations of SoS software architectures by automatically

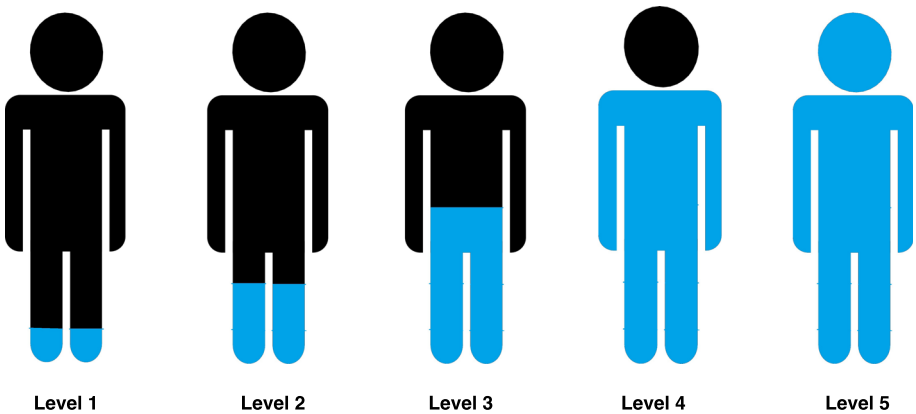

Fig. 6 Water level with a human dummy

producing stimuli generators. As such, we claim that to be reliable, a simulation must reproduce the conditions under which a SoS operates, considering both its surrounding environment (such as rain and temperature) and constituents' operational conditions (such as battery level and GPS location). Then, our evaluation is based on the success of our approach to support automatic production of stimuli generators that can (i) reproduce the surrounding environment and constituents' operational conditions and (ii) maintain the simulation running until the end of data input. Considering that we use a software architecture description as the basis to produce three different types of stimuli generators. If the software architecture is faithfully described and the generation method is correct, the stimuli generators created will address our claims. Then, we established the following research question: *How is it possible to automatically obtain a functional stimuli generator that reproduces environmental conditions to the simulation of a SoS*? To answer this question, we established a model-based approach that produces such stimuli generators from a SoS software architecture description and established the following goal to the case study (that matches our first research question). From this general goal, we derived the following research questions with their respective metrics:

RQ1: Are the (automatically created) stimuli generators functional?

Rationale. This question establishes whether or not the stimuli generators automatically generated are functional, that is, if they can work into the context of a simulation after deployed, exactly how they were created, without any manual intervention or modification.

Metric: success fee: percentage of data correctly delivered to the correspondent constituent, considering the amount of that data that is intended to be delivered.

RQ2: Is the stimuli generator capable of triggering a simulation correctly?

Rationale. The simulation only starts when the correct stimuli are received by the constituents and they start their operation, making the entire SoS operate. This research question evaluates if the simulation starts correctly.

Metric: efficiency: A participant observes if the simulation is successfully triggered by the stimuli received during its entire execution cycle.

RQ3: Is the stimuli generator capable of supporting an entire simulation execution correctly?

Rationale. The aim of a stimuli generator is supporting a simulation with a continuous emission of stimuli that keep the simulation running.

Metric: number of problems during simulation execution: given by the proportion of errors during simulations compared to the total execution of the simulation.

Research instruments

We used a FMSoS to collect all data used in the simulation. We adopted Eclipse Modeling Framework (EMF) as the platform to develop SoSADL models based on Xtext. Xtend is the transformation language, MS4ME[16] is the simulation platform, and DEVS (in particular, a DEVS dialect called DEVS) is the formalism for running the generated simulation models.

Data preparation

We obtained data collected by the sensors that are under supervision of a Brazilian entity responsible for monitoring natural disasters (Brazilian Center for Monitoring and Warnings of Natural Disasters - CEMADEN) [59]. These data were parsed and stored in a text file. Stimuli generator are fed with them, emitting them to the simulation, stimulating it until the end of the execution. The input of data triggers the constituents operation, cause their interoperability, reach the gateway, are processed creating new data that correspond to positive or negative flood alerts. We also collect these data to analyze results.

We chose a large sample of data collected by the real FMSoS from November 23, 2015, to December 31, 2015. This interval was important because during these months a number of floods occurred. This enabled us to establish whether or not our simulation results in a diversity of situations. We sent 1000 samples for each sensor, being sent every 5 min, and 1000 for crowdsourcing systems. Considering that we only had data to feed sensors in a simulation, we adapted them so to have similar data for stimuli generators for crowdsourcing systems and drones. For crowdsourcing systems, the aforementioned scale was used to classify risk between 0 and 6. So we could simulate how people would react and behave due to the changes in water level registered before by sensors. Then, we created a dataset correspondent to the data used to feed sensors. This dataset is available externally[17]. For drones, we used 5000 drone data, since the drone receives every 500 m a measurement and 2500 m flown every 5 min, totaling this amount for the entire days that we consider in our sample.

Analysis procedures of collected data

Stimuli-SoS approach is concerned of the automatic production of stimuli generators. Hence, we need to evaluate if the stimuli generators automatically produced (i) conform to an expected structure of a DEVS model that send stimuli to a simulation and (ii) are functional, correctly delivering data to the respective constituents that wait for their stimuli. Thus, a quantitative analysis can be adopted to (i) measure the correctness and similarity of stimuli generators to the expected form of a functional DEVS atomic model that deliver data, (ii) evaluate if the stimuli

generator keep its operation, delivering data along the entire simulation cycle, and (iii) evaluate if the simulation is correctly triggered and maintained in operation until the end of the input procedure. Hence, we adopted a quantitative analysis in our case study [57]. We follow a systematic approach divided in well-established steps, reporting the collection and measurement of pre-defined expected data, observing and measuring the scenario via simulation according to pre-defined metrics, and drawing conclusions from these results to answer the research questions established.

Reporting

We report our results based on the steps systematically followed to achieve the derivation of the stimuli generators for FMSoS constituents. A video shows a summary of the entire procedure[18].

1. *SoS software architecture specification*

The specification of the software architectural description of the FMSoS was conceived as a joint work between SofTware ARchitecture Team (START/ICMC) at University of São Paulo and ArchWare (IRISA) at University of South Britanny, in France. Such specification was conducted by a team of four people using SoSADL language. This step was accomplished after 2 months of work and received four iterations to perform refinements on the SoSADL syntax, to cover some gaps that were not identified before and to refine the software architectural description itself until reaching an acceptable format. We specified an FMSoS architecture with 42 sensors, 9 crowdsourcing systems, and 18 drones, as described in Fig. 3. Such specification was validated by a peer-review procedure composed by the SoSADL creator and other SoS experts. The complete SoSADL architecture specification is available externally[19].

2. *Automatic derivation of stimuli generators*

After the accomplishment of the first step, the automatic derivation step was conducted. The software architectural description produced in step 1 was used as input for this step, being processed by the model transformation script, delivering a stimuli generator for sensors that compose the FMSoS. At this step, a distinct stimuli generator is produced for each distinct type of constituent. In FMSoS case, three types of stimuli generator are conceived: one stimuli generator for sensors, another one for crowdsourcing systems, and another for drone system. The transformation runs and delivers the code in 2 s. The products of this activity (the stimuli generators themselves) were evaluated using the metrics defined in RQ1 (similarity and correctness).

Figure 7 illustrates how an automaton is derived from SoSADL system specification to create a functional stimuli generator. In DEVS, transitions can occur due to (i) a data received, expressed as `?data`, (ii) a data sent, expressed as `!data`, and (iii) a spontaneous transition, without any input or output. This is the approach we used to generate atomic models for each one of the constituent types[20]. In turn, derivation of the stimuli generator is quite different. In SoSADL, there is a special type of connection called **environment**, which abstracts interaction of an SoS with the surrounding environment, emitting outputs to the environment, or receiving stimulus from it, e.g., when the system is a sensor, as shown in the code available in Fig. 7. Some parts are hidden since they do not influence in the discussion of stimuli generation derivation. It is possible to see that the gate `energy` offers two environment connections (Lines 12 and 13): one to receive a

```
system Sensor( lps:Coordinate ) is {

  gate measurement is {
    environment connection sense is in { RawData }
    connection pass is in { MeasureData }
    connection measure is out { MeasureData }
  }

  gate energy is {
    environment connection threshold is in { Energy }
    environment connection power is in { Energy }
  }

  gate location is {
    environment connection coordinate is out {Coordinate }
  }
}
```

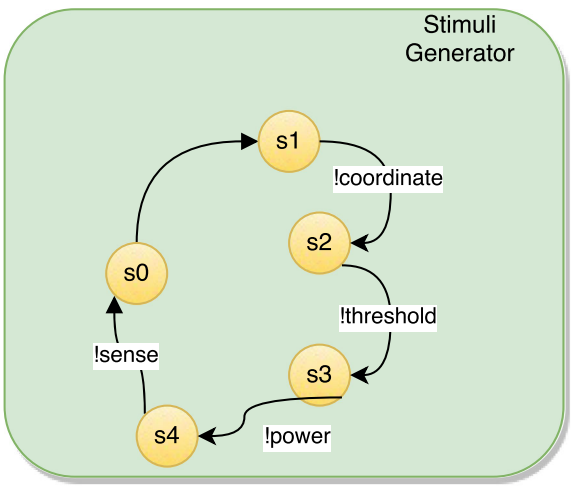

Fig. 7 Illustration of how an automaton is derived from SoSADL system specification to create a functional stimuli generator

`threshold` (a limit of energy that is considered enough to keep the sensor in operation) and `power` that is used to receive the level of battery available. Connection `sense` is that one responsible to receive raw data, i.e., the water level from that is being measured from the river by sensor actuators. Lastly, connection `coordinate` receives GPS coordinate from the sensor GPS.

SoSADL models are analyzed by the transformation algorithm, searching for environment connections. Each connection specified as an `environment` connection (underlined in Fig. 7) produces one transition in the specification of the state diagram in the resulting stimuli generator. Hence, the stimuli generator consists of a special type of model that has a continuous behavior (a behavior materialized as a loop) to emit stimuli by output state transitions, starting and maintaining the SoS operation. Figure 7 depicts a state diagram equivalent that is created with state transitions created to deliver each of one of the connection data types underlined. It delivers the aforementioned data and comes back to the state `s0`, forming a loop that continuously offer stimuli for SoS simulation. Order is not important, as constituents are only triggered when the data received matches the input data expected in the state transition in which its operation is at that moment.

Listing 2 DEVS code for a stimuli generator.

```
1  generates output on coordinate!
2  generates output on threshold!
3  generates output on power!
4  generates output on sense!
5
6  to start hold in s1 for time 1!
7  from s0 go to s1!
8  after s1 output coordinate!
9  from s1 go to s2!
10 hold in s2 for time 1!
11 after s2 output threshold!
12 from s2 go to s3!
13 hold in s3 for time 1!
14 after s3 output power!
15 from s3 go to s4!
16 after s4 output sense!
17 from s4 go to s0!
```

Listing 2 shows the code in DEVS that specifies part of the stimuli generator produced using our approach. The stimuli generator is created not only with the automaton that guides its operation, but also with specification of ports, data types, and all the apparatus necessary to make it executable and to enable the execution of the target simulation (some parts are hidden for the reader convenience). In listing 2, the stimuli generator has four output ports (lines 1 to 4) that delivers the collection of the geographic positions (coordinate), power threshold, power level (battery energy), and the water level sensed by sensors.

3. *Inclusion of stimuli generators in the target simulation model*

After the automatic derivation, the stimuli generator must be deployed in the simulation code specified in DEVS and already deployed in MS4ME environment. This step consists of moving the stimuli generator file to the simulation project in MS4ME environment. MS4ME environment automatically generates a Java file that corresponds to the execution entity of each stimuli generator. The SoS architectural description in DEVS is also adapted to include stimuli generators and to specify that they must emit data to their corresponding constituents, that is, those that hold environment connections that were used as input to produce the respective stimuli generators. Figure 3 illustrates an example of FMSoS architecture during simulation. Mediators enable transmission of data received by sensors from stimuli generators until the nearest gateway. This activity was evaluated by checking if, after deployed, the simulation become executable.

4. *Simulation execution and monitoring*

The simulation took for 6 h and 20 min (6.20 h) in Processor Intel core i5-3230M 2.60 GHz (×64), with 4 GB of RAM Memory, HD of 1 TB, and running Ubuntu 16.04 64 bits. The data corresponds to 38 days of monitoring data from the Monjolinho River. Data were stored in text files and delivered by the stimuli generators along the FMSoS, feeding the simulation. This step was evaluated according to the metrics established in research questions two and three (efficiency and number of problems during simulation execution).

We established *a four-window strategy* implemented at the gateways that receive data from constituents to confirm floods. For each four data that subsequently arrives, the gateway checks it by pairs (three possible combination of pairs of four data that arrives). If at least one pair that arrived have both their depth levels equals to or major than 100 cm (the threshold established for that city), a flood alarm is triggered. Experts remarked that one sensor could trigger a false alarm due to the possibility of sediment accumulation, which can increase the measured collected in a location, but that does not represent a flood. Hence, taking pairs was considered a valid strategy. Table 2 illustrates an example. It corresponds to real data that arrived sequentially at the gateway. Each four data that arrive are chronologically ordered, and pairs of data given by (S2,S3), (S1,S3), and (S3,S4) are analyzed. If at least one of the pairs has two measures equal or greater than 100 cm, a flood is confirmed. We did allow the sum of four measures that generate false alarms (for example, S1 = 90 cm, S2 = 90 cm, S3 = 90 cm, S4 = 130 cm). This can represent an increasing in the level of

Table 2 A sample of data sent by sensors

Sensor	Timestamp	Depth (cm)	Alert
S2	2015-11-23 01:58	58	NO
S1	2015-11-23 02:03	56	
S3	2015-11-23 02:03	54	
S4	2015-11-23 02:03	57	

water, but not a flood. Subsequent measures will confirm if it is an actual flood or not. After all the data were analyzed, we compared our results to the original results to evaluate the confidence of the automatically generated simulation.

However, it is possible to remark in Table 2, data do not arrive in order. Hence, if a flood occurs, S1 will be the first to increase its measure, followed by S2, S3, and S4. Thus, a false negative can occur due to the delay to arrive at the gateway and possible losses of data. One possible situation occurs when only one transmit data with more than 100 cm to the gateway, because it was the last one in that sequence of four measures, but the other measures, even if not 100 cm yet, can have already slightly increased, indicating a possible a flood coming. To avoid this, a new test was done: we added both measures of the other combinations that were not checked in the first cycle to avoid false diagnostics. For example, considering Table 2, we obtain the values of depth from (S2 + S3), (S2 + S4), and (S1 + S4).

After the simulation terminated, we analyzed the perceptions from the observation and answered the research questions, as follows. Figure 8 shows the biggest averages of depth of water reached in each of the days analyzed. Considering that the four-window strategy exhibits a threat of flood, days November 23 and December 21 and 30 are the most relevant. Other moments exhibit values bigger than 100 cm, but only as a momentary occurrence. The graphic enables us to analyze that the stimuli generator was capable of delivering the data continuously during all the simulation execution. Next, we discuss the answers to the research questions.

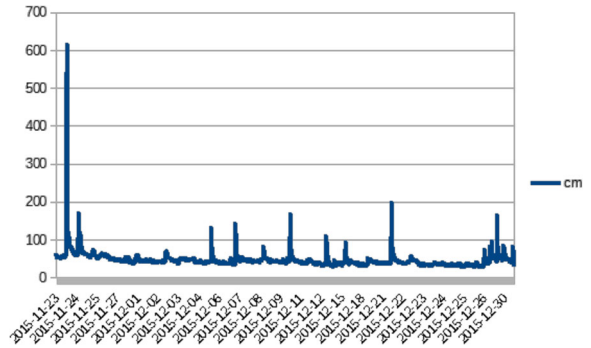

Fig. 8 Monitoring depth of water in simulation during data processing

RQ1: Are the (automatically created) stimuli generators functional?

Yes. The stimuli generators were analyzed by a specialist that agreed that it contains all necessary structures to deliver the expected behavior. Moreover, we observed their behavior during the simulation execution, and the data that arrived in the gateways. Of the data, 100% were correctly delivered to the simulation.

RQ2: Are the stimuli generators capable of triggering a simulation correctly?

Yes. We followed the entire cycle of operation of the simulation. The stimuli generators were capable of receiving the input data from the database and generating the expected stimuli for the constituents, triggering the SoS interoperability. Hence, the stimuli generator was well succeeded. Three types of stimuli generators were derived from the specifications in SoSADL, as available externally[21]. For all of them, they were able to receive data stored in text files and deliver them to the simulation.

RQ3: Is the stimuli generator capable of supporting an entire simulation execution correctly?

During the entire cycle, the generator stimulated the simulation, completing the operation cycles of this SoS. The group of samples were grouped in four measures each to give a flood alert or not, depending on the analysis of the data. We followed the same strategy implemented in the real gateway: from four data that arrives, if two are above the threshold, the flood alert is triggered. Twenty-nine flood alerts occurred along some hours of flood (considering that each group of four data received that whose sum was more than the threshold triggered the alert). During the considered period, besides one effective flood (November 23), in which the level of water arrived at almost 7 m, two other real threats of flood occurred on December 7 and 21. With no failures, the stimuli generator was capable of supporting the simulation during its entire cycle of operation.

Discussion

Our solution promotes the automatic production of stimuli generators from architectural descriptions of SoS. It can create a distinct stimuli generator for each distinct type of constituent that forms a SoS. We applied the same methodology to produce three different types of functional stimuli generators: one for a smart sensor, one for crowdsourcing system, and another one for a drone, diversifying our data sources and characterizing the required multiple source of data of a case study.

Considering our context, requirement elicitation for Flood Monitoring SoS was a joint effort between several institutions, such as ICMC/USP, IRISA/UBS, CEMADEN, and Franhoufer Institute. They elaborated the requirements, describing the surrounding environment for such SoS. After that, we used this document as an input to

create the architectural description in SoSADL. Despite the environment being highly dynamic, a SoS is concerned only with a subset of the possible stimuli that can be received. Stimuli set completeness is a SoS requirement engineering issue. Considering a Flood Monitoring SoS as an example, data used in our example (such as water level) are relevant and enough to draw conclusions about a possible flood and the respective flood alert (as this is the intended purpose for this SoS). Dynamics of the environment is handled by the sensors actuators themselves (at hardware level). Oscillations in values are passed to software-level, received, and processed to draw actions. Then, we believe that the success of our approach relies on the effectiveness of a SoS to deal with the set of environment data delivered to it (even if this data set is specific and restrict). Moreover, at simulation level, we can observe how SoS will behave considering the specific data received. If we notice that data delivered are not sufficient to draw conclusions, a new study can be conducted to increase data expected in order to increase precision of floods detection. However, considering the data set that we worked on and these types of constituents, SoS was able to detect all the flood threats that effectively occurred. Hence, we can conjecture that the stimuli set completeness is acceptable.

We also addressed scale, heterogeneity, and autonomy, which are important concerns for SoS. About the scale, we run an example with 69 constituents, without considering gateways, mediators, and drone bases. For all constituent types, stimuli generators were automatically derived and worked correctly. About the heterogeneity, we used five different types of constituents. About autonomy, all of these constituents exhibit their own structure, behavior, and independent operation.

It is important to highlight how much the adoption of stimuli generators reduces the manual work of the SoS simulation. To perform this work manually, considering that each of the stimuli would consist of a click, each click demands a reasoning from the human analyst. If each click needs 10 s to be decided and executed by a human, in a sample of 1000 data for each crowdsourcing system, 1000 for each sensor, and 5000 for each drone, this would result in an amount of 141,000 samples to be entered by the human user into an architecture of 42 sensors, 9 crowdsourcing systems, and 18 drones. Therefore, this work would require 1,410,000 s, which amounts to almost 392 h of work, or almost 50 days of work in 8-h days. Our procedure needed little more than 6 h to run.

Contributions

The contributions of our approach are listed as follows:

- *Productivity*: We claim that Stimuli-SoS contributes to the productivity in the SoS engineering. Using our approach, we simulated 38 days in little more than 6 h. The effort necessary to correctly simulate the activities of a human to reproduce real data accordingly would be significantly larger than using our solution, as discussed earlier. Thus, our approach is almost 65 times more productive than a manual approach, considering the architecture we used.
- *Reuse*: Programming the model transformation to automatically produce a stimuli generator by one specialist with integral dedication took 5 days of work (a total of approximately 40 h). Despite the learning curve associated to DEVS modeling, Xtend programming, and domain-specific knowledge to adapt model transformations, the model transformation can be reused in a myriad of other domains. Producing a stimuli generator for each type of constituent of a SoS takes the same amount of time. The same specialist that produced the model transformation also produced an operational stimuli generator to realize the final format that should be achieved.
- *Model-based engineering for SoS*: Application of model-based methods in SoS engineering is still at the beginning [10]. Moreover, a recent study reveals that MBE has been adopted for the development of SoS (around 60% of included studies in a systematic mapping applied MBE for development of SoS [43]). MBE has been applied in SoS context for managing systems complexity, developing candidate architectures, and verifying design decisions early in the development process. Thus, we believe that our approach contributes to SoS software engineering by establishing a novel model-based approach to support SoS simulation and environment modeling. The automatic generation of stimuli generator for simulation of software architectures of SoS purposes is a contribution for Software Engineering for SoS and SoSE, as these techniques were broadly adopted for hardware benchmarking, but rarely applied in software engineering, in particular, software engineering for SoS;
- *Environment modeling*: Environment modeling is an emerging issue, not only for simulation domain or SoS domain, but also for modern software engineering as a whole [60, 61]. It is important to improve techniques and methods to model the surrounding environment in which systems will be deployed, predicting situations that could not be dealt with effectively without this type of modeling and preventing failures not envisioned before. These are vital issues as SoS becomes increasingly autonomous and ubiquitous, working on domains such as flood monitoring [2] and crisis management [62].
- *Stimuli-SoS workflow*: Stimuli generators are produced using a SoS software architecture

description as input, following well-defined systematic steps that achieve the production of functional stimuli generators deployed in a simulation. The proposed workflow is also a contribution of our work, as it exhibits potential to be reproduced in other scenarios and contributes by prescribing how to produce this type of simulation structure.

Threats to validity

Considering conclusion validity, we conjecture that it is not a remarkable threat for this study, since we do not have a statistical strength in our conclusions and we do not compare our approach with others, but use an exploratory study to draw our conclusions and justify our claims. Considering internal validity, we can raise the strategy to divide the data received by the gateway in a four-window strategy. As this is the number of constituents, we do not perform remarkable partitions in data. Hence, we consider that this is not a significant threat. Considering construction validity, we draw our conclusions based on an approach that was systematically followed to automatically derive stimuli generators. Hence, we more observe than we measure. Further quantitative studies must be carried out to compare other forthcoming generations for different domains and that one we worked on here. Considering external validity, we run a case study in which, using approach, three different types of stimuli generators were produced, each one for a different type of system: a crowdsourcing system, a drone, and a sensor. As such, we increased our sources of evidence, even considering that all of them work in the same single simulation. Further investigation must be carried out, but there is some potential to application in other domains and generalization.

Regarding other threats, we can mention the possibility of failures if the SoS architect does not qualify the environment connections in SoSADL with the keyword `environment`. If it occurs, simulation can fail because expected input can be never received. Indeed, any error regarding the declaration of environment connections at design time can affect the final simulation. Moreover, more accurate evaluation in larger contexts and applications are still required. Our approach was evaluated in regards to its success to support automatic production of stimuli generators that can correctly (i) reproduce the surrounding environment and constituents operational conditions. Considering that we use a software architecture description as the basis to produce stimuli generators, if the software architecture is not faithfully described, the stimuli generators created can not be correctly produced. We relieved this threat by submitting the software architecture description to a specialist. Another threat to validity is the correctness of the model transformation. To minimize the impact of this threat, a specialist conducted a manual inspection and carefully evaluated if each transformation rule produced exactly the expected output considering each input given.

Related work

Recent studies have investigated the adoption of simulation in software engineering [63], and simulation has certainly been applied for SoS development [16, 36, 64]. Additionally, initiatives have invested in the simulation of software architectures, but not specifically for software architectures of SoS, such as SySML [65], MatLab/Simulink [66], Palladio[22] [67], Bogado et al. [36], and Alexander et al. [68]. Other initiatives invested in modeling and simulating SoS, but with no support for software architecture concept [15].

The development of stimuli generators for simulation purposes is not a new trend [35, 37, 38]. Initiatives have investigated the adoption of stimuli generators for hardware benchmarking. For example, Al-Hashimi [37] describes the use of stimuli generators to produce digital input signals for simulation purposes of analogic-digital systems. Kitchen and Kuehlmann present an approach to stimulate simulations of hardware with a stimuli generator that performs a random generation of input stimuli that obey a set of declaratively specified input constraints. Rahman and Lombigit [33] describe the development of a software that systematically generates stimulus required for code simulation (functional and timing) of new digital processors in gamma spectroscopy system. Yang et al. [35] adopts simulations for verification purposes to evaluate the correctness of system-on-chips. They apply stimuli generator to offer a broader coverage of test cases aiming to confirm the correctness of the chip operation. Thus, they do not work on top of software architectures, automating only the generation of the stimuli but not the infrastructure that will forward stimuli to the simulation.

For simulations in the context of SoS software engineering and software architecture, only few works have investigated stimuli generators. Table 3 compares the closest related approaches considering the following six characteristics addressed by our approach:

1. Description of SoS software architectures : Does the highlighted approach adopt some formalism to describe SoS software architectures?

Table 3 Comparison between co-related approaches

Approach	1	2	3	4	5	6
DEVS [15]	No	No	Yes	Yes	Yes	No
Kewley et al. [69]	No	No	No	No	No	No
Soyez [70]	No	No	Yes	No	No	No
Stimuli-SoS	Yes	Yes	Yes	Yes	Yes	Yes

2. Simulation of SoS software architectures: Does the approach support simulation of SoS software architectures?
3. Environment modeling: Does this approach adopt some type of environment modeling for simulation purposes?
4. Environment simulation: Does the approach adopt some type of environment simulation?
5. Adoption of stimuli generator: Does the approach adopt stimuli generator as the technique to inject inputs into the simulation?
6. Automatic derivation of stimuli generator: Does the approach prescribe some type of automatic derivation or mechanisms to stimulate a simulation?

DEVS is a well-established formalism for simulating SoS in virtual environments [15]. DEVS deals with the system architecture, i.e., a simulation model in DEVS considers software and hardware aspects of all the constituents that compose a SoS, and for the SoS itself. DEVS takes into account several important characteristics of software architectures, such as data types, constituent systems (represented as atomic models), constituent behaviors (expressed as labeled input diagrams), SoS dynamics, and how constituent exchange data (coupled models), events, and the overall organization of such constituents. However, it does not preserve the architectural details of SoS software architecture specification and relies on a low-level abstraction formalism, as discussed before.

Kewley et al. claim that constituents should be simulated by isolated simulations and that such simulations should be *federated*, that is, they should interoperate in a synergistic way to form the whole simulation of a SoS [69]. They adopt a framework called SySHub to play the role of *glue* that enables federations of models to support SoS simulation. However, they do not work on the level of software architecture (simulation or representation), despite the fact that they consider distributed interactive simulation (DIS) and high-level architecture (HLA) as potential architectural and interoperability methods for description and federated simulations of SoS [71]. However, even these notations do not tackle the concepts we address in our approach related to software architecture. They consider environment modeling as a potential forthcoming contribution of the SySHub system. However, we did not find continuity at this research topic or more recent papers that report supporting environmental modeling in SySHub context. Therefore, automatic derivation of stimuli generator is not currently covered in that approach.

Soyez et al. propose an agent-based tool to support modeling of static and dynamic aspects of SoS [70]. Their formalism is based on the multi-level agent-based model IRM4MLS, which allows the representation of multiple entities that can interoperate at different levels, i.e., a constituent can be itself an SoS, hence supporting different levels of granularity [72]. To evaluate their approach, they implemented a co-simulation of a directed SoS coping with a reconfiguration problem in the domain of intelligent autonomous vehicles. Despite the use of co-simulation, they do not provide any evidence of concerns with the notion of software architecture, nor automatic code generation or stimuli generators. They support the modeling of environment and claim that their formalism is suitable for simulation. However, there is no evidence strengthening their claim.

Considering these previous works, there is a gap regarding the automatic derivation of stimuli generators based on software architectural descriptions of SoS. Our approach bridges these gaps and contributes by advancing the state of the art about simulation of software architectures of SoS. The next section discusses threats to validity.

By the nature of SoS, environments are only partially known at design time [8]. It is important to emphasize that our approach is to generate stimuli for simulation, not to automatically create the data to be used in the simulation. A prototype of the data is created that is functional, but there is no technique for creating data that is reliable to reality. Currently, this type of data is collected from other sources, and inserted via Java code into the body of the stimulus generator. Nonetheless, there is an important contribution towards environmental modeling in SoS engineering. In this stage of the contribution, we automatically create a virtual entity for the simulation capable of delivering the data in a rhythmic rhythm, imitating the surrounding environment from the data provided to the stimulus generator to feed the simulation. In a next step, we intend to invest in the automatic creation of these data by a more accurate description of the environment. The next section brings final remarks and potential for future research.

Final remarks and forthcoming steps

This article presented Stimuli-SoS, an approach to systematically and automatically derive stimuli generators to support the execution of simulation of SoS. We established the following research question to be answered: *How is it possible to automatically obtain a functional stimuli generator that reproduces environmental conditions to the simulation of a SoS*? We concluded that the stimuli generators automatically created:

1. conform with the expected format. The transformation derived is what was expected;
2. were capable of receiving input data from the database and generating expected stimuli for the constituents, triggering the SoS;
3. were capable of correctly supporting an entire simulation execution; and

4. reproduce the environmental conditions of an SoS to become simulations functional without manual intervention.

Potential applications and forthcoming investigation can be conducted relying on the advances produced by our research. Co-simulation, for instance, is an important but significant challenge. It exhibits the potential to establish a communication between industrial simulators. However, even for the context of simulation of single subsystems that compose a whole monolithic system, co-simulation is still a matter of investigation [41, 73]. Stimuli generators have the potential to be the interface that enables receiving the injection of values from industrial simulators. The automatic derivation of these stimuli generators from software architectural descriptions of SoS with support for environment modeling may enhance the fidelity of the code generated and the proximity with the environmental modeling provided by industrial simulators.

Simulations have been recognized as source of empirical evidences for software engineering [63]. Hence, the adoption of our approach can leverage the research on empirical software engineering supported by simulations. Adopting our approach can aid in the automation of simulation-based studies, deriving stimuli generators to be applied during the simulation operation.

Stimuli generators materialize an infrastructure to support Verification, Validation, and Testing (VV&T) activities in an automated way [74]. They can be applied to benchmark a SoS, working as a platform for VV&T of SoS. Each transition in an atomic model can work as a test case, and data can be provided by external files that hold test cases that are automatically generated by a testing tool [74, 75]. Moreover, VV&T for SoS is currently a challenging research issue and point of investigation in software engineering for SoS [43].

Our approach also exhibits a potential to become an architectural pattern for modeling of simulations. As stimulating a simulation is a recurrent problem, we can establish a stimuli generator as a systematic and repetitive solution that can be adapted according to the context in which it will be applied.

Simulation is an important branch of software engineering for SoS. It exhibits a remarkable potential to be largely adopted in software engineering for SoS in the forthcoming years. Then, investigating potentials of automation in the coverage of tests and correctness of operation is paramount to avoid damages, losses, and financial problems that could be brought by an SoS deployed with errors. We believe that our approach can contribute to leverage the degree of trustworthiness delivered by an SoS.

Endnotes

[1] Throughout this manuscript, SoS will be used interchangeably to express singular and plural. Moreover, for the context of this text, SoS denotes software-intensive SoS.

[2] http://www.omg.org/spec/UML/2.5/

[3] http://www.omg.org/spec/SysML/1.4/

[4] http://www.compass-project.eu/

[5] http://danse-ip.eu/home/

[6] Mediators are architectural elements that establish communication between two or more constituents [76]

[7] Additional details about the syntax of architecture descriptions in SoSADL can be found in [8].

[8] https://www.mathworks.com/products/simulink.html

[9] http://www.omgsysml.org/

[10] SPEM - Software & Systems Process Engineering Metamodel: http://www.omg.org/spec/SPEM/

[11] https://goo.gl/vPbKcL

[12] http://www.ana.gov.br/

[13] http://www.cprm.gov.br/

[14] Credits for the images used to compose the figure: http://goo.gl/TTOlAa, http://goo.gl/QCUAKY, http://goo.gl/a9Y0Dw, https://goo.gl/rFkYJ6, https://goo.gl/8YojYj, https://goo.gl/XyWEZw, https://goo.gl/VpftdV, https://goo.gl/dfMPLl.

[15] Moreover, our example scenario also covers constituents heterogeneity, autonomy, and SoS scale, characteristics that are commonly assigned to SoS, as well [78].

[16] http://www.ms4systems.com/pages/ms4me.php

[17] http://www.inf.ufg.br/~valdemarneto/journalMaterials/stimuli-sos.html

[18] https://vimeo.com/220144774

[19] https://goo.gl/xk5h3z

[20] We do not discuss this mechanism with details in this paper, since the focus is the representation and derivation of a stimulus generator. Other details are discussed in a forthcoming paper.

[21] https://goo.gl/xk5h3z

[22] http://www.palladio-simulator.com/

Additional file

Additional file 1: Additional listings for the reader convenience. (PDF 136 kb)

Abbreviations
ADL: Architectural description language; CEMADEN: Brazilian center for monitoring and warnings of natural disasters; DEVS: Discrete event systems specification; DEVSNL: Discrete event systems specification natural language; EMF: Eclipse modeling framework; FMSoS: Flood monitoring system-of-system; GPS: Global positioning system; HLA: High-level architecture; IRISA: Institut de Recherche en Informatique et Systèmes Aléatoires; M&S: Modeling and simulation; MATLAB: MATrix LABoratory; MBE: Model-based engineering; MS4ME: *Modeling and Simulation* Modeling Environment; SoS: System-of-system; SosADL: Systems-of-systems architectural description language; SPEM: Software & systems process engineering metamodel; SySML: Systems modeling language; UML: Unified modeling language; VV&T: Verification, validation and test; V&V: Verification and validation

Acknowledgements
We thank Prof. Dr. Leslie Foulds (INF/UFG) for the language review; and Dr. Flavio Horita (CEMADEN) by providing data that we use and adapt to serve as input to our stimuli generators.

Funding
Núcleo de Informação e Coordenção do br NIC.br for the waiver received.

Availability of data and materials
The data we used to be processed during the simulation is not public. Hence, we can not make it available. However, we believe that all the necessary data to understand our findings are available at the body of this manuscript.

Authors' contributions
VVGN contributed to all sections. CEP created figures and wrote excerpts to all sections. LG contributed to background section and evaluation. MG contributed to evaluation and review of the entire article. WM contributed by running a new case study, models specification, and textual reporting. FO contributed by analyzing case study results. EYN contributed in the entire manuscript, besides supervising this research. All authors read and approved the final manuscript.

Author's information
VVGN - Permanent Software Engineering lecturer at Informatics Institute of Universidade Federal de Goiás, Goiânia, Brazil. He is also a PhD candidate in a co-tutelle program between University of São Paulo (São Carlos, Brazil) and Université de Bretagne-Sud (Vannes, France). He received his bachelor's degree (2009) and Msc (2012) in Computer Science from Universidade Federal de Goiás, Goiânia, Brazil. In 2015, he was one of the general chairs of XI Brazilian Symposium on Information Systems (SBSI, in cooperation with ACM), in Goiânia, Brazil. Currently, he is a member of the Special Committee of Information Systems of the Brazilian Computer Society (CESI/SBC), and also an SBC Associate and ACM Member. His research interests are focused on software-intensive systems-of-systems, model-based software engineering, software architectures, and simulation.
CEP - PhD in Computer Science by ITA, full professor at Pontifical University of São Paulo, and post-doc research at University of São Paulo.
LG - PhD candidate at University of São Paulo at São Carlos.
MG - PhD candidate at University of São Paulo at São Carlos.
WM - Undergraduate student at University of São Paulo at São Carlos.
FO - Full professor of Computer Science (holding a Research Excellence Award from the Ministry of Higher Education and Research of France) serving as Research Director at the UMR CNRS IRISA, in Britany, France. He received his BEng from ITA, Sao José dos Campos, SP, Brazil, and his MSc, PhD and HDR from the University of Grenoble, France. He has published over 200 refereed journal and conference papers and has been editor of over 15 journal special issues and research books. He has served on programme committees of over 100 international conferences, e.g., ICSE, ESEC/FSE, has chaired more than 10 of them, of which the French, European, and IEEE/IFIP International Conferences on Software Architecture (CAL, ECSA, ICSA). His research interests are centred on formal languages, processes, and tools to support the efficient architecture of complex software-intensive systems and systems-of-systems.
EYN - MS (1998) and PhD (2006) in Computer Science from the University of São Paulo (USP), Brazil. She conducted her post-doctoral in 2011–2012 in Fraunhofer IESE, Germany, and in 2014–2015 at University of South Brittany, France. She is associate professor in the Department of Computer Systems at University of São Paulo, Brazil. Her main research interests are software architecture, reference architectures, systems-of-systems, software testing, and evidence-based software engineering. She is a member of the IEEE and SBC (Brazilian Computer Society).

Competing interests
The authors declare that they have no competing interests.

Publisher's Note

Author details
[1]University of São Paulo, Av. Trabalhador Sancarlense, 400, 13566-590 São Carlos, Brazil. [2]University of South Brittany, Rue André Lwoff, 56000 Vannes, France. [3]Universidade Federal de Goiás, Alameda das Palmeiras, 74690-900 Goiânia, Brazil. [4]Pontifical University of São Paulo, R. Monte Alegre, 05014-901 São Paulo, Brazil.

Received: 26 January 2017 Accepted: 28 September 2017
Published online: 13 October 2017

References
1. Maier MW (1998) Architecting principles for systems-of-systems. Syst Eng 1(4):267–284
2. Oquendo F (2016) Software architecture challenges and emerging research in software-intensive systems-of-systems. In: 10th European Conference on Software Architecture. Springer, Copenhagen. pp 3–21
3. Guessi M, Oquendo F, Nakagawa EY (2016) Checking the architectural feasibility of systems-of-systems using formal descriptions. In: 2016 11th System of Systems Engineering Conference (SoSE). IEEE, Kongsberg. pp 1–6
4. Graciano Neto VV, Paes CEB, Oquendo F, Nakagawa EY (2016) Supporting simulation of systems-of-systems software architectures by a model-driven derivation of a stimulus generator. In: Proceedings of Workshop on Distributed Development, Software Ecosystems and Systems of Systems, ser. WDES' 16. SBC, Maringá. pp 61–70
5. ROAD2SOS (2013)) Road2sos project - roadmaps for systems-of-systems engineering. http://road2sos-project.eu/cms/front_content.php. Accessed June 2016
6. Nielsen CB, Larsen PG, Fitzgerald J, Woodcock J, Peleska J (2015) Systems of systems engineering: basic concepts, model-based techniques, and research directions. ACM Comput Surv 48(2):18:1–18:41. [Online]. Available: http://doi.acm.org/10.1145/2794381
7. Chiprianov V, Falkner K, Szabo C, Puddy G (2014) Architectural support for model-driven performance prediction of distributed real-time embedded systems of systems. In: Avgeriou P, Zdun U (eds). 8th European Conference on Software Architecture, ser. ECSA. Springer, Vienna. pp 357–364
8. Oquendo F (2016) Formally describing the software architecture of systems-of-systems with SosADL. In: 11th IEEE System of Systems Engineering Conference (SoSE). IEEE, Kongsberg. pp 1–6
9. Cerrudo C (2015) Keeping smart cities smart: preempting emerging cyber attacks in U.S. cities. Tech Rep
10. Steinhogl W (2015) Trustworthy systems of systems—a prerequisite for the digitalization of industry. ERCIM News 2015(102):1–2
11. Nami M, Suryn W (2013) Software trustworthiness: past, present and future. In: Yuan Y, Wu X, Lu Y (eds). Trustworthy Computing and Services: International Conference, ISCTCS 2012, Beijing, China, Revised Selected Papers. Springer Berlin Heidelberg, Berlin. pp 1–12
12. Graciano Neto VV, Oquendo F, Nakagawa EY (2016) Systems-of-systems: challenges for information systems research in the next 10 years. In: Proceedings of Big Research Challenges in Information Systems in Brazil (2016–2026) at Brazilian Symposium on Information Systems, ser. GRANDSI-BR/SBSI. SBC, Florianópolis. pp 1–3
13. Wachholder D, Stary C (2015) Enabling emergent behavior in systems-of-systems through bigraph-based modeling. In: 2011 6th International Conference on Systems of Systems Engineering (SoSE). IEEE, San Antonio. pp 334–339
14. Mittal S, Rainey L (2015) Harnessing emergence: the control and design of emergent behavior in system of systems engineering. In: SCS. SCSI, San Diego. pp 1–10

15. Zeigler BP, Sarjoughian HS, Duboz R, Souli J-C (2012) Guide to modeling and simulation of systems of systems. Springer, Berlim
16. Graciano Neto VV, Guessi M, Oliveira LBR, Oquendo F, Nakagawa EY (2014) Investigating the model-driven development for systems-of-systems. In: Proceedings of the 2014 European Conference on Software Architecture Workshops, ser. ECSAW '14. ACM, Vienna. pp 22:1–22:8
17. Vangheluwe H (2008) Foundations of modelling and simulation of complex systems. Electron Commun EASST 10:1–12
18. Sanchez-Montanes MA, Konig P, Verschure PFMJ (2002) Learning sensory maps with real-world stimuli in real time using a biophysically realistic learning rule. IEEE Trans Neural Netw 13(3):619–632
19. Rajkumar RR, Lee I, Sha L, Stankovic J (2010) Cyber-physical systems: the next computing revolution. In: Proceedings of the 47th Design Automation Conference, ser. DAC '10. ACM, New York. pp 731–736. [Online]. Available: http://doi.acm.org/10.1145/1837274.1837461
20. Selic B (2012) MDE basics with a UML focus. In: Proceedings of 12th International School on Formal Methods for the Design of Computer, Communication and Software Systems: Model-Driven Engineering. Bertinoro. p 1. talk available at: www.sti.uniurb.it/events/sfm12mde/slides/selic.pdf. Accessed 8 Oct 2017
21. Hehenberger P, Vogel-Heuser B, Bradley D, Eynard B, Tomiyama T, Achiche S (2016) Design, modelling, simulation and integration of cyber physical systems: methods and applications. Comput Ind 82:273–289. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0166361516300902
22. France R, Rumpe B (2007) Model-driven development of complex software: a research roadmap. In: 2007 Future of Software Engineering, ser. FOSE '07. IEEE Computer Society, Washington. pp 37–54. [Online]. Available: http://dx.doi.org/10.1109/FOSE.2007.14
23. Carle P, Kervarc R, Cuisinier R, Huynh N, Bedouët J, Rivière T, Noulard E (2012) Simulation of systems of systems. AerospaceLab 4:1–10. [Online]. Available https://hal.archives-ouvertes.fr/hal-01184315. Accessed 8 Oct 2017
24. Baldwin WC, Sauser B, Cloutier R (2015) Simulation approaches for system of systems: events-based versus agent based modeling. Procedia Comput Sci 44:363–372. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050915002689
25. Falkner K, Szabo C, Chiprianov V, Puddy G, Rieckmann M, Fraser D, Aston C (2016) Model-driven performance prediction of systems of systems. Softw Syst Model:1–27. [Online]. Available: http://dx.doi.org/10.1007/s10270-016-0547-8. Accessed 8 Oct 2017
26. Banks J (1999) Introduction to simulation. In: Proceedings of the 31st Conference on Winter Simulation: Simulation—a Bridge to the Future - Volume 1, ser. WSC '99. ACM, New York. pp 7–13. [Online]. Available: http://doi.acm.org/10.1145/324138.324142
27. Bosch J (2000) Design and use of software architectures: adopting and evolving a product-line approach. Addison-Wesley, New York
28. Santos DS, Oliveira BMG, Oquendo F, Delamaro M, Nakagawa EY (2014) Towards the evaluation of system-of-systems software architectures. In: Proceedings of Workshop on Distributed Development, Software Ecosystems and Systems of Systems, ser. WDES. SBC, Maceió. pp 53–57
29. Chalmers DJ (2006) Strong and weak emergence. In: Davies P, Clayton P (eds). The Re-Emergence of Emergence. Oxford University Press, Oxford
30. Choi BK, Kang D (2013) Modeling and simulation of discrete event systems, 1st ed. Wiley Publishing, Mapo-Gu Seoul
31. BKCASE Editorial Board (2017) The Guide to the Systems Engineering Body of Knowledge (SEBoK), v. 1.8. R.D. Adcock (EIC). Hoboken, NJ: The Trustees of the Stevens Institute of Technology. Accessed DATE. www.sebokwiki.org. BKCASE is managed and maintained by the Stevens Institute of Technology Systems Engineering Research Center, the International Council on Systems Engineering, and the Institute of Electrical and Electronics Engineers Computer Society. http://sebokwiki.org/wiki/Guide_to_the_Systems_Engineering_Body_of_Knowledge_(SEBoK). Accessed 8 Oct 2017
32. Bruneau J, Consel C Diasim: a simulator for pervasive computing applications. Softw Pract Experience 43(8):885–909. [Online]. Available: http://dx.doi.org/10.1002/spe.2130
33. Rahman NAA, Ramli AR, Lombigit L, Abdullah NA, Khalid MAH (2014) Stimulus generation technique for code simulation of FPGA based gamma spectroscopy system. In: ADVANCING NUCLEAR RESEARCH AND ENERGY DEVELOPMENT: Proceedings of the International Nuclear Science, Technology & Engineering Conference 2013 (iNuSTEC2013), vol 1584, no. 1. AIP Publishing, Melville. pp 77–83
34. Piccolboni L, Pravadelli G (2014) Simplified stimuli generation for scenario and assertion based verification. In: 2014 15th Latin American Test Workshop - LATW. IEEE, Fortaleza. pp 1–6
35. Yang S, Wille R, Große D, Drechler R (2012) Coverage-driven stimuli generation. In: 2012 15th Euromicro Conference on Digital System Design. IEEE, Cesme. pp 525–528
36. Bogado V, Gonnet S, Leone H (2014) Modeling and simulation of software architecture in discrete event system specification for quality evaluation. Simulation 90(3):290–319. [Online]. Available: http://dx.doi.org/10.1177/0037549713518586
37. Al-Hashimi B (1995) The art of simulation using PSpice: analog and digital, 1st ed. CRC Press, Inc., Boca Raton
38. Kitchen N, Kuehlmann A (2007) Stimulus generation for constrained random simulation. In: Proceedings of the 2007 IEEE/ACM International Conference on Computer-aided design, ser. ICCAD '07. IEEE Press, San Jose. pp 258–265. [Online]. Available: http://dl.acm.org/citation.cfm?id=1326073.1326127
39. Plaza SM, Markov IL, Bertacco V (2007) Toggle: a coverage-guided random stimulus generator. In: Proc. International Workshop on Logic and Synthesis (IWLS). IEEE, San Diego. pp 351–357
40. Barton PI, Pantelides CC (1994) Modeling of combined discrete/continuous processes. AIChE J 40(6):966–979. [Online]. Available: http://dx.doi.org/10.1002/aic.690400608
41. Gomes C (2016) Foundations for continuous time hierarchical co-simulation. In: ACM Student Research Competition at MODELS. CEUR, Saint Malo. pp 7–13
42. Graciano Neto VV, Guessi M, de Oliveira LBR, Oquendo F, Garcés L, Nakagawa EY (2015) A conceptual map of model-driven development for systems-of-systems. In: Proceedings of Workshop on Distributed Development, Software Ecosystems and Systems of Systems, ser. WDES' 15. SBC, Belo Horizonte. pp 89–92
43. Lana CA, Souza NM, Delamaro ME, Nakagawa EY, Oquendo F, Maldonado JC (2016) Systems-of-systems development: initiatives, trends, and challenges. In: Proceedings of XLII Conferencia Latinoamericana de Informá, ser. CLEI '16. IEEE Press, Valparaiso. pp 1–10
44. Sendall S, Kozaczynski W (2003) Model transformation: the heart and soul of model-driven software development. IEEE Softw 20(5):42–45. [Online]. Available: http://dx.doi.org/10.1109/MS.2003.1231150
45. Bettini L (2013) Implementing domain-specific languages with Xtext and Xtend. Packt Publishing, Birmingham
46. Eclipse (2012) Acceleo. [Online]. Available: http://www.eclipse.org/acceleo/. Accessed 8 Oct 2017
47. Sun Y, Demirezen Z, Lukman T, Mernik M, Gray J (2008) Model transformations require formal semantics. In: Lawall J, Réveillère L (eds). Domain-Specific Program Development. ACM, Nashville. p 5
48. ISO/IEC/IEEE (2011) Systems and software engineering - Architecture description. International Organization for Standardization (ISO)/International Electrotechnical Commission (IEC)/Institute of Electrical and Electronics Engineers (IEEE), Geneva. ISO/IEC/IEEE 42010
49. Guessi M, Graciano Neto VV, Bianchi T, Felizardo KR, Oquendo F, Nakagawa EY (2015) A systematic literature review on the description of software architectures for systems of systems. In: Proceedings of ACM Symposium on Applied Computing, ser. SAC, Salamanca. pp 1433–1440
50. Foster H, Mukhija A, Rosenblum DS, Uchitel S (2011) Rigorous software engineering for service-oriented systems: results of the SENSORIA Project on software engineering for service-oriented computing. Springer, Berlin. ch. Specification and Analysis of Dynamically-Reconfigurable Service Architectures, [Online]. Available http://dx.doi.org/10.1007/978-3-642-20401-2_20. Accessed 8 Oct 2017
51. Allen R, Garlan D (1997) A formal basis for architectural connection. ACM Trans Softw Eng Methodol 6(3):213–249. [Online]. Available: http://doi.acm.org/10.1145/258077.258078
52. Oquendo F (2004) π-ADL: An architecture description language based on the higher-order typed π-calculus for specifying dynamic and mobile software architectures. SIGSOFT Softw Eng Notes 29(3):1–14. [Online]. Available: http://doi.acm.org/10.1145/986710.986728
53. Oquendo F (2016) π-Calculus for SoS: A foundation for formally describing software-intensive systems-of-systems. In: 11th IEEE System of

Systems Engineering Conference (SoSE). IEEE, Kongsberg. pp 1–6. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7542925&isnumber=7542882
54. Hu J, Huang L, Cao B, Chang X (2014) Extended DEVSML as a model transformation intermediary to make UML diagrams executable. In: 26th International Conference on Software Engineering and Knowledge Engineering (SEKE). Knowledge Systems Institute Graduate School, Vancouver
55. Bass L, Clements P, Kazman R (2012) Software architecture in practice. Addison-Wesley Longman Publishing Co., Inc., Boston
56. Cavalcante E, Batista TV, Oquendo F (2015) Supporting dynamic software architectures: from architectural description to implementation. In: The Working IEEE/IFIP Conference on Software Architecture (WICSA) 2015, Montreal. pp 31–40. [Online]. Available: http://dx.doi.org/10.1109/WICSA.2015.21. Accessed 8 Oct 2017
57. Runeson P, Höst M (2009) Guidelines for conducting and reporting case study research in software engineering. Empirical Softw Engg 14(2):131–164
58. de Albuquerque JP, Horita FEA, Degrossi LC, dos Santos Rocha R, de Andrade SC, Restrepo-Estrada C, Leyh W (2017) Leveraging volunteered geographic information to improve disaster resilience: lessons learned from AGORA and future research directions. In: Volunteered Geographic Information and the Future of Geospatial Data. IGI Global. pp 158–184
59. Horita FE, de Albuquerque JP, Degrossi LC, Mendiondo EM, Ueyama J (2015) Development of a spatial decision support system for flood risk management in Brazil that combines volunteered geographic information with wireless sensor networks. Comput Geosci 80:84–94
60. David O, Lloyd II JAW, Green T, Rojas K, Leavesley G, Ahuja L (2013) A software engineering perspective on environmental modeling framework design: the object modeling system. Environ Model Softw 39:201–213. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1364815212000886. thematic Issue on the Future of Integrated Modeling Science and Technology
61. Iqbal MZ, Arcuri A, Briand L Environment modeling and simulation for automated testing of soft real-time embedded software. Softw Syst Model 14(1):483–524
62. Santos DS, Oliveira BRN, Duran A, Nakagawa EY 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, Pittsburgh. pp 304–309
63. de França BBN, Travassos GH (2016) Experimentation with dynamic simulation models in software engineering: planning and reporting guidelines. Empir Softw Eng 21(3):1302–1345. [Online]. Available: http://dx.doi.org/10.1007/s10664-015-9386-4
64. Xia X, Wu J, Liu C, Xu L (2013) A model-driven approach for evaluating system of systems. In: International Conference on Engineering of Complex Computer Systems (ICECCS). IEEE, Singapore. pp 56–64
65. OMG Sysml open source specification project. Accessed June 2016 http://sysml.org/
66. MATLAB (2010) version 7.10.0 (R2010a). The MathWorks Inc., Natick. https://www.mathworks.com/products/matlab.html
67. Becker S, Koziolek H, Reussner R (2009) The Palladio component model for model-driven performance prediction. J Syst Softw 82(1):3–22. [Online]. Available: http://dx.doi.org/10.1016/j.jss.2008.03.066
68. Alexander P, Nicolaescu A, Lichter H (2015) Model-based evaluation and simulation of software architecture evolution. In: Proceedings of International Conference on Software Engineering Advances, ser. ICSEA, Barcelona. pp 153–156
69. Kewley R, Cook J, Goerger N, Henderson D, Teague E (2008) Federated simulations for systems of systems integration. In: 2008 Winter Simulation Conference. IEEE, Miami. pp 1121–1129
70. Soyez J-B, Morvan G, Merzouki R, Dupont D (2014) A multilevel agent-based approach to model and simulate systems of systems. In: InTraDE project final workshop, Lille
71. IEEE Standard for Modeling and Simulation (M amp;S) High Level Architecture (HLA)—Framework and Rules. IEEE Std 1516-2010 (Revision of IEEE Std 1516-2000), pp 1–38
72. Morvan G, Veremme A, Dupont D (2011) IRM4MLS: the influence reaction model for multi-level simulation. Springer, Berlin
73. Schweizer B, Lu D, Li P (2016) Co-simulation method for solver coupling with algebraic constraints incorporating relaxation techniques. Multibody Syst Dyn 36(1):1–36. [Online]. Available: http://dx.doi.org/10.1007/s11044-015-9464-9
74. Anand S, Burke EK, Chen TY, Clark J, Cohen MB, Grieskamp W, Harman M, Harrold MJ, McMinn P, et al (2013) An orchestrated survey of methodologies for automated software test case generation. J Syst Softw 86(8):1978–2001
75. Korel B (1990) Automated software test data generation. IEEE Trans Softw Eng 16(8):870–879
76. Wiederhold G (1992) Mediators in the architecture of future information systems. Computer 25(3):38–49
77. Graciano Neto VV (2017) A model-based approach towards the building of trustworthy software-intensive systems-of-systems. In: Proceedings of the 39th International Conference on Software Engineering, ICSE 2017 - Companion Volume. IEEE, Buenos Aires. pp 425–428
78. Jamshidi M (2009) System of systems engineering—innovations for 21st century, ser. Wiley Series in Systems Engineering and Management. Wiley