Validating Emergent Behaviors in Systems-of-Systems through Model Transformations

Valdemar Vicente Graciano Neto

University of São Paulo, São Carlos, Brazil
IRISA-UMR CNRS/Université de Bretagne-Sud, Vannes, France
Universidade Federal de Goiás, Goiânia, Brazil
valdemarneto@usp.br

Abstract

Systems-of-Systems (SoS) are formed by independent systems termed as constituents. SoS exhibit dynamic properties called emergent behaviors, which are a global functionality that appears as a result of the interoperability among constituents. However, software architecture descriptions of SoS are often static. In turn, dynamic models such as simulation models (also adopted to specify SoS) do not use to preserve software architecture details, which can hamper the software quality. In this paper, we propose a model transformation approach to harmonize software architecture descriptions of SoS and simulation models to support validation of emergent behaviors. We model a software architecture of SoS by the adoption of SosADL, a novel architectural description language (ADL) for SoS, and transform it to DEVS, a formalism for simulation of SoS. Our approach offers a dynamic view to architectural descriptions of SoS, preserving the architectural integrity of the SoS, and supporting the visualization and validation of emergent behaviors. We evaluate our proposal through a case study conducted within the context of a real SoS in operation for flood monitoring in an urban area. Preliminary results show that the transformation is feasible, generating functional simulation models that support the validation of emergent behaviors.

Categories and Subject Descriptors Software and its engineering [*Software system models*]: Model-driven software engineering

Keywords Systems-of-Systems, Emergent Behaviors, Model Transformations, Software Architecture.

1. Motivation and Research Problem

Systems-of-Systems (SoS)[1] are a set of independent, heterogeneous constituent systems that form a larger system in order to accomplish a set of missions, that is, functional goals assigned to the SoS as a whole (Maier 1998). They are likely to form the next generation of software-intensive systems (Jamshidi 2008; Boehm 2006), often supporting missions in critical domains, such as emergency and crisis response in smart cities (ROAD2SOS 2013).

One remarkable feature of SoS is its software architecture. Software architectures play a fundamental role in software quality (Nakagawa et al. 2013). Thus, it is prevalent to adopt models that capture precisely such software architectures (Nielsen et al. 2015). However, besides structure and individual systems behaviors, SoS exhibit a dynamic property called emergent behavior (Maier 1998; Boardman and Sauser 2006; Cavalcante et al. 2014; Mittal and Rainey 2015). This property emerges at runtime as a result of the interoperability of the constituent systems (Maier 1998). However, this property has not been captured by modern architectural description languages (ADL) (Michael et al. 2009, 2011; Guessi et al. 2015b). The operation and success of an SoS is intrinsically associated with the emergent behaviors and its underlying software architecture. Thus, an approach to support validation of emergent behaviors in SoS software architectures is prominent and still an issue (Zave 1993; Sauser et al. 2010; Nielsen et al. 2015).

Under another perspective, the Systems Engineering (SE) community has developed approaches based on simulations to support dynamic properties for SoS (Michael et al. 2009; Sauser et al. 2010; Zeigler et al. 2012; Mittal and Rainey 2015; Wachholder and Stary 2015). However, SE approaches do not properly support software architecture description (Guessi et al. 2015b). Their formalisms use to rely on notations based on input/output events and state diagrams, which sacrifices the high-level of abstraction required for SoS software architecture descriptions.

Then, we pose the following research question: *How to validate emergent behaviors in SoS preserving their software architectures?* SofTware ARchitecture Team (START/ICMC-USP) and ArchWare (IRISA/UBS) research groups have worked on supporting both specification and validation of SoS software architectures. As a result, a new ADL called SosADL was proposed (Oquendo and Legay 2015; Oquendo 2016a,b). However, SosADL is not subject to simulation, yet. In this direction, this paper proposes a transformation approach from SosADL to DEVS (Discrete Event System Specification) (Zeigler et al. 2012), a simulation formalism. Our approach harmonizes both the software architecture of the SoS and a simulation perspective, supporting the validation of emergent behaviors, while still preserving the SoS software architecture description. The remainder of this paper is structured as follows. Section 2 describes foundations. Section 3 outlines the mapping from SoSADL to DEVS. Section 4 discusses our preliminary results, and Section 5 brings final remarks.

[1] For sake of simplicity, SoS will be used interchangeably to express singular and plural.

2. Foundations

SoS share five well-defined characteristics postulated by Maier (Maier 1998): (i) operational independence, since constituents operate both autonomously and in the SoS context; (ii) managerial independence of constituents, that is, distinct stakeholders and institutions can own them; (iii) emergent behavior, a synergistic behavior yielded by the constituents interoperability; (iv) evolutionary development, once the SoS evolves as a result from the evolution of its constituents; and (v) distribution, since their communication relies on some network technology.

Emergent behavior is a singular characteristic of SoS. It is triggered by the reception of stimulus and data exchanged between the constituents, and between the SoS and the environment (Graham 2013). Such behaviors are a holistic phenomena manifested after a certain number of interactions among the constituents that produces a superior result that could not be delivered by any one of them in isolation. Examples include a *home security* behavior that could emerge from a set of individual systems installed in a smart home. In short, an emergent behavior is classified, despite further refinements, as *Weak* or *Strong* (Chalmers 2006; Mittal and Rainey 2015). The former is reducible to a simulation, whilst the latter can not be simulated by a computer (for example, life as a result from the parts that constitute alive organisms). Even when working on weak emergent behaviors, unexpected behaviors can emerge, jeopardizing involved stakeholders, with potential to cause hazards and losses (Chalmers 2006). Then, it is paramount validating such behaviors, that is, checking the conformance between the pre-established missions and the respective emergent behaviors that accomplish them.

Under another perspective, software architectures correspond to the fundamental structure of a software system. They comprise software elements, relations among them, and the rationale, properties, and principles governing their design and evolution (Bass et al. 2012; iso 2011). In turn, a software architecture of an SoS is its fundamental structure, including its constituents and connections between them, properties of the constituents and of the environment (Nielsen et al. 2015). Indeed, architectural descriptions are the main work product that express a software architecture (iso 2011). They are often specified based on ADL (iso 2011).

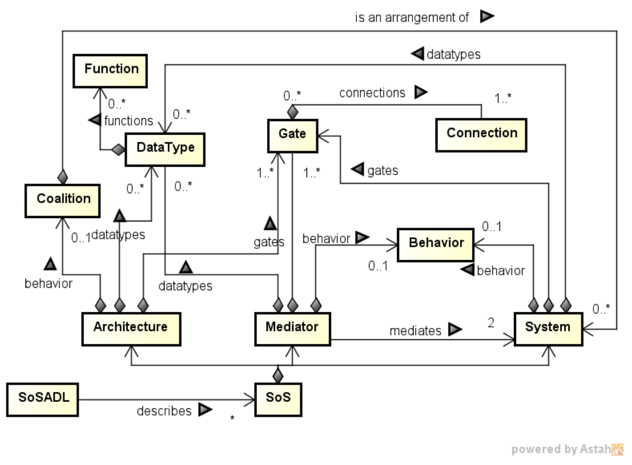

Figure 1. An excerpt of the SosADL metamodel.

SosADL is a novel ADL conceived to support the specification of architectural descriptions for software architecture of SoS. It has been created to overcome the drawbacks exhibited by precedent ones (Oquendo and Legay 2015; Oquendo 2016a,b), such as the difficulties to specify multiple individual systems interoperating, while still considering their software architecture, and to deal with constituents that are known at design-time. SosADL is formally founded on π-calculus for SoS (Oquendo 2016b), a novel formalism to describe intercommunicating processes. Figure 1 shows an excerpt of SosADL metamodel. In short, SosADL describes SoS, which can be expressed as a combination of architectures, systems, and mediators declarations. Mediators are architectural elements concerned with establishing communication between two or more constituents (Wiederhold 1992). An architecture has an intrinsic structure and behavior declaration (materialized as a coalition), data types, and gates declarations. Gates are abstractions that enable the establishment of connections. A connection can be established to receive stimulus from or act on the environment, or simply for a communication between constituents. Data types can have inherent functions, and functions can have associated expressions. Mediators and systems have gates, data types, and behaviors, as well as the SoS architecture itself. Coalitions are an arrangement of systems (constituents), and Systems are mediated by mediators (Oquendo and Legay 2015). SosADL supports specifying weak emergent behavior by the idea of coalition, a temporary alliance for combined action among constituents connected by mediators. Those behaviors are specified as part of the coalition behavior, documenting how constituents should interact to accomplish a given set of missions. Expressions and values are suppressed in this representation[2].

SosADL models are not executable yet. Hence, a dynamic view for architectural descriptions based on SosADL is still required. Model transformations are a well-accepted approach that aid software engineers to establish correspondences between models (Sun et al. 2008). In particular, Model-Driven Development techniques have been investigated in the context of SoS (Graciano Neto et al. 2014; Graciano Neto et al. 2015). Besides, a broad set of tools is available to achieve a proper level of automation using transformation tools, such as Xtend (Bettini 2013) and Acceleo (Eclipse 2012). Thus, a model transformation can be carried out between SosADL models and a simulation model in order to preserve the architectural descriptions and to achieve the dynamics required to visualize and validate emergent behaviors.

A validation approach for emergent behaviors in SoS software architectures requires a dynamic view that externalizes such behaviors. Such an approach should allow software architects to predict and validate desired emergent behaviors, and also prevent unexpected behaviors by monitoring the SoS dynamics. Currently, there are four major categories of techniques for validating software architectures[3]: scenario-based, simulation-based, mathematical/logical-based, and experience-/metric-based. Considering the dynamic nature of SoS software architectures, a simulation-based approach is undoubtedly the best choice[4]. Nonetheless, known ADL for SoS are not subject to runtime execution or simulation. Thus, they do not provide a dynamic view for SoS software architecture descriptions (Guessi et al. 2015a).

Simulation-based approaches have supported the validation of dynamic properties for SoS (Nielsen et al. 2015; Mittal and Rainey 2015; Michael et al. 2009; Sauser et al. 2010; Zeigler et al. 2012; Wachholder and Stary 2015) and have recently been investigated in the context of software engineering (de França and Travassos 2016). Such approaches (Michael et al. 2011; de França and Travas-

[2] More details about the syntax of architecture descriptions in SosADL and its elements can be found in (Oquendo 2016a).

[3] Dobrica and Niemela discuss further details about validation methods for software architecture (Dobrica and Niemele 2002; Michael et al. 2009, 2011).

[4] Nielsen et al. deeply discuss simulation approaches for SoS (Nielsen et al. 2015).

Approach	Support for Software Architectural Description of SoS	Support for Specification of Emergent Behaviors	Support for Validation of Emergent Behaviors
Bigraph (Wachholder and Stary 2015)	✗	✓	✗
Cavalcante et al. (Cavalcante et al. 2014)	✗	✓	✗
CML (Woodcock et al. 2012)	✓	✗	✗
DEVS (Zeigler et al. 2012)	✗	✓	✓
Michael et al. (Michael et al. 2011)	✗	✓	✗
Systomics (Sauser et al. 2010)	✗	✓	✗
SySML (OMG 2016a)	✓	✗	✗
UML (OMG 2016b)	✓	✗	✗
Xia et al. (Xia et al. 2013)	✗	✗	✓
Our approach	✓	✓	✓

Table 1. Comparison between co-related approaches.

sos 2016; Wachholder and Stary 2015; Xia et al. 2013): (i) support the validation of expected emergent behaviors, (ii) empower the observation of unexpected emergent behaviors; (iii) enable the prediction of errors, diagnosing them and permitting corrections; and (iv) provide a visual and dynamic view, reproducing stimuli that the system can receive from the environment.

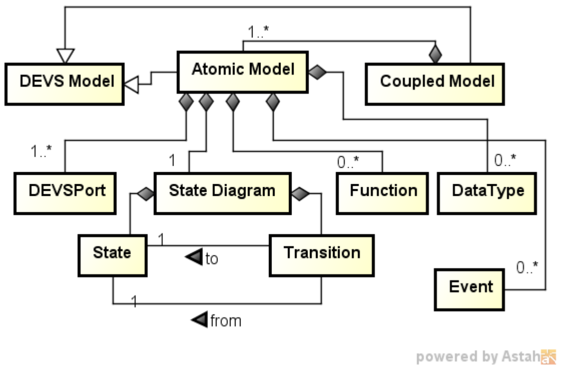

Figure 2. DEVS simplified metamodel for DEVSNL, adapted from (Cetinkaya et al. 2012).

DEVS is an option to support SoS simulation (Zeigler et al. 2012). It corresponds to a modeling formalism for SoS based on atomic and coupled models. Atomic models represent individual entities in the SoS (for instance, systems), while coupled models represent a combination of atomic models. Figure 2 depicts DEVS basic elements. Atomic models comprise the following elements: (i) ports (input and output); (ii) a labeled state diagram that performs transitions due to input or output events; (iii) functions that can be used to process data; (iv) data types; and (v) events. In particular, a state diagram is a set of states and transitions, in which a transition is necessarily from one state to another state. Coupled models are expressed as a System Entity Structure (SES), that is, a formal structure governed by a small number of axioms that expresses how atomic models communicate between themselves (Zeigler et al. 2012). We chose SosADL and DEVS to compose our approach since SosADL suppresses the disadvantages of the other ADL as aforementioned, and DEVS was conceived especially for simulation of SoS. We adopted a DEVS dialect called DEVS Natural Language (DEVSNL) that enables programming atomic and coupled models in a human-like format in tools such as MS4ME[5]. The combination of SosADL and DEVS offers the necessary level of abstraction for architectural descriptions of SoS and the required dynamic view for validation of emergent behaviors.

Related Work. Several initiatives have been proposed to address the description of SoS software architectures, representation, and validation of emergent behaviors of SoS. Table 1 shows a comparison among related work and characteristics offered by them.

Bigraph-based modeling offers a solution for mathematically representing emergent behaviors in SoS through a formalism based on a special type of graphs (Wachholder and Stary 2015). However, there is not a support for description of software architectures of SoS, nor a validation approach.

Cavalcante et al. (Cavalcante et al. 2014) report a model transformation approach that adopts π-ADL architectural models to describe dynamic architectures, that is, software architectures in which components and connectors can be created, interconnected, and/or removed during system execution, transforming π-ADL models in Go software code. Their work does not have focus on the validation of SoS' emergent behaviors, and Go is not a simulation model. Although π-ADL provides architectural description models for concurrent and communication processes, it does not provide straightforward abstractions of some particular concepts of SoS, such as mediator and coalition.

Xia et al. (Xia et al. 2013) perform evaluation of SoS architectures documented in DoDAF (dod 2010) through a model-driven approach that transforms system architecture models in executable models described in Simulink[6]. More precisely, their approach is concerned with the entire SoS, including hardware issues, while our approach is focused only on software architecture. Moreover, they focus their evaluation on technical aspects of the SoS, measuring non-functional properties such as feasibility and efficiency. Validation of the rationality and correctness of logic and behavior of the architectural models are considered supported and a possibility, but validation is not properly performed.

Another perspective involves bio-inspired initiatives, such as the Systomics approach (Sauser et al. 2010), establishing analogies to construct models to represent emergent behaviors in SoS. However, it does not properly offer a validation for emergent behaviors. In turn, Michael et al. (Michael et al. 2011) propose a mathematical notation to deal with verification and validation of software architectures of SoS (as a whole). Simulation is mentioned, focusing on verification of non-functional requirements, with no mention to validation of emergent behaviors. Moreover, all of those notations do not exhibit a suitable level of abstraction for the purposes of software architecture description of SoS.

DEVS itself is an approach for SoS simulation and validation of emergent behavior (Zeigler et al. 2012). However, it does not

[5] http://goo.gl/NmBBuu

[6] www.mathworks.com/products/simulink/

support a representation, relying on systems engineering concepts, which can be unsuitable for software architects. Transformations from distinct models to DEVS in order to make the former executable have been proposed (Cetinkaya et al. 2012; Gonzalez et al. 2015; Hu et al. 2014). However, no focus on software architectural descriptions of SoS in DEVS has been found.

In parallel, languages for specifying SoS architectures can be identified (Guessi et al. 2015b): UML (OMG 2016b) (semi-formal), SysML (OMG 2016a) (semi-formal), CML (Woodcock et al. 2012) (formal). UML does not support a suitable specification of software architectures of SoS (Guessi et al. 2015b). SysML lacks specific structures for modeling some aspects of software architecture of SoS and it is also a static specification, with no support for validation of emergent behaviors (Guessi et al. 2015b). CML is a formal language especially conceived for SoS formal specification within the context of the Comprehensive Modelling for Advanced Systems of Systems (COMPASS) alliance. However, it focuses on the verification of emergent behaviors, not on their validation (Fitzgerald et al. 2013).

3. A Transformation Approach to Support Validation of Emergent Behaviors

We established a model-based transformation approach to map SosADL models into SES/DEVS models. In our approach, we distinguish two main roles involved in SoS conception: Software Architect and System Architect. From the software architect's viewpoint, a SoS is examined under a static perspective regarding the software architecture itself without support for visualizing emergent behaviors. From the systems architect's viewpoint, the SoS is analyzed from a dynamic perspective considering state diagrams, constituents, inputs and outputs, with support for emergent behavior validation but without an explicit software architecture. To overcome the drawbacks exhibited, our approach merges the advantages offered by systems and software architecture bridging both models by means of a mapping between them, supporting both software architecture and validation of emergent behaviors for SoS software architecture, offering dynamic and static views.

Table 2 presents the correspondences between SosADL and DEVS to establish the transformation.

Table 2. Mapping between SoSADL and SES/DEVS

SoSADL	**SES/DEVS**
Architecture	Coupled Model
Behavior	State Diagram
Connection	DEVS Port
Coalition	Coupled Model
Data Type	Data Type
Function	DEVS Function
Mediator	Atomic Model
SoS	Coupled Model
System	Atomic Model

System and Mediator. Both concepts become atomic models in DEVS. They are represented as single entities in the simulation model. They have their own behavior expressed as a state diagram.
Behavior Declaration. Behaviors in SosADL are specified by a list of statements. These statements are similar to basic instructions available in programming languages, such as type declaration, and sequences. Each statement becomes one or more transitions in a a state diagram (automaton) in DEVS. Important operations in SosADL include **send** and **receive**, that are translated into state transitions that cause the emission of data to another system or transitions that are triggered by the reception of data, respectively.
Architecture, Coalition, and SoS. These structures are mapped into coupled models, being converted in the specification of how they promote the SoS operation.
Connections and Gates. In SosADL, connections are abstractions of links between gates from different communicating entities that establish a channel to transmit data. Multiple connections (input and output), can be established on the same gate. Translating to SES/DEVS, the gate concept is suppressed, and connections are mapped into ports (input and output ports).
Data Types. Data types in SosADL become data types in DEVS. They can be Abstract Data Types (ADT) or simple types, such as integer. They can be exchanged in messages among systems.
Functions. Analogously, function declarations of SosADL are converted in functions in DEVS. The discussion of their mapping is beyond the scope of this paper.

During the transformation, a SoS architectural description written in SosADL is verified against the abstract syntax of SosADL described in Xtext[7]. If the SosADL code conforms to the Xtext grammar, the code is submitted as input for an Xtend[8] script that materializes the code generator. A functional code written in DEVSNL is generated as output.

Listing 1. A transformation rule corresponding to `receive` statement in SosADL, which becomes an `input` transition in DEVSNL.

```
1  def String generationOfInputTransition()'''
2    passivate in s<<fromState>>!
3    when in s<<fromState>> and receive <<
         dataReceived>> go to s<<toState>>!
4  '''
```

Listing 1 shows an excerpt of a transformation rule used to map a statement `receive` written in SosADL in an input transition specification in DEVS. This rule is in the context of an ADT defined within the Xtend transformation code. This ADT is an abstraction of a DEVS Transition and gives access to the states involved in the transition (source and target), and to the data being received. The automata is extracted using Xtend and constructed according to the abstract syntax tree provided by Xtext/Xtend. Then, these data are used to create a couple of lines of code in DEVS. Line 2 in Listing 1 expresses that the execution waits in the state s_n (where n is the number that identifies an specific state) until receiving a specific data. Then, Line 3 declares that when this occurs, a transition to a second state occurs. Line 4 closes the scope of the method specified in Xtend.

4. Preliminary Results

We adopted our approach in a practical case study: a Flood Monitoring SoS (FMSoS), that is, an SoS composed of smart sensors that use software to monitor the occurrences of floods in an urban area of the city of São Carlos in Brazil. Rivers cross the city and, when the rains are intense, floods often occur, causing property loss, damage, and serious danger to the population. The FMSoS is used to illustrate the validation of a single emergent behavior: *flood alert*. Constituents are smart sensors. They are spread on the river edges at a regular distance with mediators between them. The data is collected by sensors and transmitted until reaching the gateway. When flooding occurs, the gateway emits an alarm for the public authorities.

[7] https://eclipse.org/Xtext/

[8] xtend-lang.org/

The transformation was successfully run and the atomic and coupled models accordingly generated. A brief video demonstrating the generated simulation is available[9]. Some important points must be highlighted about this specific transformations, as follows:

- **Adaptations in source model.** Specification of such a transformation helped in the improvement of the source model, since some conflicting names of variables were diagnosed during the transformation, good practices of specification for SosADL models were discovered, and even incomplete specification became evident when the transformation started to be executed, aiding in the completion of the source code. Hence, a possible conjecture is that the adoption of model transformations in our approach may have helped in the improvement of the quality of a source model of SoS, since this approach offers a visualization of the resultant dynamics, and the consequences of decisions in the specification of the source model;
- **Maturity of the language.** A model-driven approach for specification and execution of software architectures of SoS helped even in the maturity of the SosADL language itself. Since the ADL is quite novel and still subject to adaptations, some issues about the possibility of multiple behaviors in a same constituent were discussed and some points about the definition of data types were also clarified, such as the convention of assign the type Integer for any type of data that were specified as abstract (such as a type `Energy`), but without an explicit attribute type assigned to be effectively used.
- **Stimuli Generator.** It is important to highlight that there exist atomic models in the functional example called stimuli generators. They consist of abstractions of systems that continuously emit the stimuli necessary to trigger the SoS operation, such as delivering the coordinates `lps` originally delivered by a GPS, or the level of water collected by a sensor. Stimuli generator is used to create the first two portions of code, sending `lps` coordinates, `powerLevel`, and a first `sense`, that is, a data collected by the sensor that will trigger systems behavior execution in all of the systems. We also automatically derive it from the specification of the source models written in SosADL. This structure makes the simulation functional. More details about this approach are found in (Graciano Neto et al. 2016).
- **Transformations execution.** Each transformation takes no more than milliseconds to be run. They run without errors. Two different types of transformations are executed. One to generate atomic models, and another to generate coupled models. Hence, we run the transformation four times: one to generate sensor, another one to generate mediator, another one to generate gateway (all atomic models), and a last one to generate the coupled model. This is necessary because the structures that model an architecture in SosADL do not hold definitions about the data types being transferred (essential for coupled models). Thus, it is necessary to infer the type being transferred between atomic models to generate it automatically in the coupled models. The strategy adopted was: (i) generation of all the atomic models, (ii) saving the definition of the connections in the SosADL specification (format `connection::gate;dataTypeName`) in a text file and (iii) reading it during the coupled model generation. Since the name of the pair `connection::gate` is unique for the entire SoS, it is possible to infer the type of the data being transferred.

[9] `https://goo.gl/cf6sef`

5. Final Remarks

This paper presented preliminary results of a research being conducted to answer this question: *How to validate emergent behaviors in SoS preserving their software architectures?* We proposed a solution to transform a software architecture description written in SosADL to a simulation model in DEVS, harmonizing both formalisms by means of a mapping between them. Our approach contributes by: (i) offering a solution as how to transform an software architecture description of SoS into a simulation formalism, guaranteeing traceability between those models, high-level of abstraction in the SoS software architecture specification and automation; (ii) supporting validation of weak emergent behaviors by analyzing the conformance between SoS missions and emergent behaviors manifested during SoS operation; and (iii) offering a visual and dynamic view for SoS software architecture descriptions.
Uniqueness. Our approach is the only one reported that provides a dynamic view for SoS software architectural descriptions and validation support for emergent behaviors by means of a transformation. SosADL supports representation of dynamic properties such as emergent behaviors. Furthermore, as a consequence, model transformations provide the operational semantics for SosADL models, that is, the meaning of SosADL statements in terms of execution, translating it in simulation models (Sun et al. 2008), such as (Shakshuki et al. 2015).
Contributions. Specification of an SoS in SosADL is more abstract and leads to programming in a language closer to software architect's current practices. Indeed, SosADL is the unique formal current state of the art notation for describing software architectures of SoS that deals with emergent behaviors (Guessi et al. 2015b; Oquendo 2016a). Moreover, our approach provides some contribution (directly or indirectly) to some of the topics of interest for the MODELS community, such as:

- **Development and use of models**: providing an operational semantics by means of a model transformation is a quite novel approach. Moreover, we establish this for an ADL that describes software architectures of SoS. As far as we know, we do not know other initiatives in this direction;
- **Integration of modeling languages and tools (hybrid multi-modeling approaches)**: Our approach harmonizes distinct paradigms of model-based engineering (software engineering and systems engineering) by means of transformation. It is not a new trend (Sun et al. 2008; Shakshuki et al. 2015), but combining a software architecture model for SoS and a simulation model is unique and a novel approach.
- **Modeling with, and for, new and emerging systems and paradigms**: Software-intensive SoS is an emergent domain. A recent systematic literature review revealed that about 75% of the publications addressing SoS appeared in the last five years and approximately 90% in the last 10 years (Guessi et al. 2015b; Oquendo and Legay 2015), whilst most of those studies communicates open issues on SoS software architecture and engineering. Software Engineering of SoS (SESoS), in turn, is even more recent and subject to investigation. Hence, we believe our solution represents an advance towards a more accurate treatment of emergent behaviors in SoS software architectures;

Future works include (i) further case studies and experiments, (ii) the extension of our approach to validate multiple concurrent emergent behaviors, and (iii) a deeper investigation on the emergence of unpredicted behaviors, establishing techniques to prevent them.

Acknowledgments

The author thanks CNPq (grant:201230/2015-1/SWE) and FAPESP (grant: 2013/20317-9) for the financial support.

References

The DoDAF Architecture Framework Version 2.02. US. Department of Defense, Aug. 2010.

ISO/IEC/IEEE Systems and software engineering – Architecture description. *ISO/IEC/IEEE 42010:2011*, pages 1–46, Dec 2011.

L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. Addison-Wesley Professional, 3rd edition, 2012.

L. Bettini. *Implementing Domain-Specific Languages with Xtext and Xtend*. Packt Publishing, 2013.

J. Boardman and B. Sauser. System of systems - the meaning of of. In *SOSE*, pages 6 pp.–, Los Angeles, California, USA, April 2006.

B. Boehm. A view of 20th and 21st century software engineering. In *Proceedings of the 28th International Conference on Software Engineering*, ICSE '06, pages 12–29, New York, NY, USA, 2006. ACM.

E. Cavalcante, F. Oquendo, and T. V. Batista. Architecture-Based Code Generation: From π-ADL Architecture Descriptions to Implementations in the Go Language. In *ECSA*, pages 130–145, Vienna, Austria, 2014.

D. Cetinkaya, A. Verbraeck, and M. D. Seck. Model Transformation from BPMN to DEVS in the MDD4MS Framework. In *STMS*, pages 1–6, Orlando, USA, 2012.

D. J. Chalmers. Strong and weak emergence. In P. Davies and P. Clayton, editors, *The Re-Emergence of Emergence*. Oxford University Press, 2006.

B. B. N. de França and G. H. Travassos. Experimentation with dynamic simulation models in software engineering: planning and reporting guidelines. *Empirical Software Engineering*, 21(3):1302–1345, 2016.

L. Dobrica and E. Niemele. A survey on software architecture analysis methods. *IEEE Trans. Softw. Eng.*, 28(7):638–653, July 2002.

Eclipse. Acceleo, 2012. URL `http://www.eclipse.org/acceleo/`.

J. Fitzgerald, S. Foster, C. Ingram, P. G. Larsen, and J. Woodcock. Model-based Engineering for Systems of Systems: the COMPASS Manifesto. Technical Report Manifesto Version 1.0, COMPASS Interest Group, October 2013. URL `http://www.compass-research.eu/Project/Publications/MBESoS.pdf`.

A. Gonzalez, C. Luna, M. Daniele, R. Cuello, and M. Perez. Towards an automatic model transformation mechanism from UML state machines to DEVS models. *CLEI Electron. J.*, 18(2), 2015.

V. V. Graciano Neto, M. Guessi, L. B. R. Oliveira, F. Oquendo, and E. Y. Nakagawa. Investigating the model-driven development for systems-of-systems. ECSAW '14, pages 22:1–22:8, New York, NY, USA, 2014. ACM.

V. V. Graciano Neto, M. Guessi, L. B. R. de Oliveira, F. Oquendo, L. Garcs, and E. Y. Nakagawa. A conceptual map of model-driven development for systems-of-systems. WDES' 15, pages 89–92, Belo Horizonte, Brazil, 2015. SBC.

V. V. Graciano Neto, C. E. B. Paes, F. Oquendo, and E. Y. Nakagawa. Supporting simulation of systems-of-systems software architectures by a model-driven derivation of a stimulus generator. WDES' 16, pages 1–10, Maringá, Brazil, 2016. SBC.

B. Graham. *Nature's Patterns - Exploring Her Tangled Web*. FreshVista, 2013.

M. Guessi, E. Cavalcante, and L. B. R. Oliveira. Characterizing architecture description languages for software-intensive systems-of-systems. In *3rd SESoS*, pages 12–18, Florence, Italy, May 2015a. IEEE.

M. Guessi, V. V. Graciano Neto, T. Bianchi, K. R. Felizardo, F. Oquendo, and E. Y. Nakagawa. A systematic literature review on the description of software architectures for systems of systems. In *SAC*, pages 1433–1440, Salamanca, Spain, April 2015b.

J. Hu, L. Huang, B. Cao, and X. Chang. Extended DEVSML as a Model Transformation Intermediary to Make UML Diagrams Executable. In *SEKE*, 2014.

M. Jamshidi. System of systems - innovations for 21st century. In *2008 IEEE Region 10 and the Third international Conference on Industrial and Information Systems*, pages 6–7, Dec 2008.

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

J. B. Michael, R. Riehle, and M. T. Shing. The verification and validation of software architecture for systems of systems. In *SoSE*, pages 1–6, Albuquerque, NM, USA, May 2009.

J. B. Michael, D. Drusinsky, T. W. Otani, and M.-T. Shing. Verification and validation for trustworthy software systems. *IEEE Software*, 28(6): 86–92, 2011.

S. Mittal and L. Rainey. Harnessing Emergence: The Control and Design of Emergent Behavior in System of Systems Engineering. In *SCS*, pages 1–10, San Diego, CA, USA, 2015. SCSI.

E. Y. Nakagawa, M. Gonçalves, M. Guessi, L. B. R. Oliveira, and F. Oquendo. The state of the art and future perspectives in systems of systems software architectures. SESoS '13, pages 13–20, New York, NY, USA, 2013. ACM.

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

OMG. SysML Open Source Specification Project, 2016a. Available in: http://sysml.org/. Last Access: June 2016.

OMG. UML: Unified Modeling Language, 2016b. http://www.omg.org/spec/UML. April 2016.

F. Oquendo. Formally Describing the Software Architecture of Systems-of-Systems with SosADL. In *SOSE*, pages 1–6, Kongsberg, Norway, June 2016a. IEEE.

F. Oquendo. π-Calculus for SoS: A Foundation for Formally Describing Software-intensive Systems-of-Systems. In *SOSE*, pages 7–12, Kongsberg, Norway, June 2016b. IEEE.

F. Oquendo and A. Legay. Formal Architecture Description of Trustworthy Systems-of-Systems with SosADL. *ERCIM News*, (102), 2015.

ROAD2SOS. Road2SoS Project - Roadmaps for Systems-of-Systems Engineering, 2013. `http://road2sos-project.eu/cms/front_content.php`. Last Access: July 2016.

B. Sauser, J. Boardman, and D. Verma. Systomics: Toward a Biology of System of Systems. *IEEE Trans. on Systems, Man, and Cybernetics*, 40 (4):803–814, 2010.

E. Shakshuki, S. Galland, A.-U.-H. Yasar, N. Messaoudi, A. Chaoui, and M. Bettaz. An operational semantics for uml 2 sequence diagrams supported by model transformations. *Procedia Computer Science*, 56: 604 – 611, 2015.

Y. Sun, Z. Demirezen, T. Lukman, M. Mernik, and J. Gray. Model Transformations Require Formal Semantics. In J. Lawall and L. Réveillère, editors, *Domain-Specific Program Development*, page 5, Nashville, United States, 2008.

D. Wachholder and C. Stary. Enabling emergent behavior in systems-of-systems through bigraph-based modeling. In *SOSE*, pages 334–339, San Antonio, TX, USA, May 2015. IEEE.

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

J. Woodcock, A. Cavalcanti, J. Fitzgerald, P. Larsen, A. Miyazawa, and S. Perry. Features of CML: A formal modelling language for Systems of Systems. In *SOSE*, pages 1–6, Genova, Italy, July 2012.

X. Xia, J. Wu, C. Liu, and L. Xu. A model-driven approach for evaluating system of systems. In *ICECCS*, pages 56–64, Singapore, July 2013.

P. Zave. Feature interactions and formal specifications in telecommunications. *Computer*, 26(8):20–29, Aug. 1993.

B. P. Zeigler, H. S. Sarjoughian, R. Duboz, and J.-C. Souli. *Guide to Modeling and Simulation of Systems of Systems*. Springer, 2012.