Research-Team ArchWare

Software Architecture

IRISA Site: Vannes

Activity Report

2016

1 Team

Head of the team
Flavio Oquendo, Full Professor, PEDR, Université de Bretagne-Sud

Administrative assistant
Sylviane Boisadan, BIATSS, Université de Bretagne-Sud

Members of the team Isabelle Borne, Full Professor, Université de Bretagne-Sud
Nicolas Belloir, Assistant Professor, Ecoles de St-Cyr Coëtquidan
Jérémy Buisson, Assistant Professor, Ecoles de St-Cyr Coëtquidan
Régis Fleurquin, Associate Professor, HDR, Université de Bretagne-Sud
Elena Leroux, Assistant Professor, Université de Bretagne-Sud
Salah Sadou, Associate Professor, HDR, PEDR, Université de Bretagne-Sud
Gersan Moguérou, Research Engineer, Université de Bretagne-Sud

Visitors Thais Batista, Associate Professor, University of Rio Grande do Norte - UFRN (Brazil)
Jair Leite, Associate Professor, University of Rio Grande do Norte - UFRN (Brazil)
Elisa Nakagawa, Associate Professor, University of Sao Paulo - USP (Brazil)

Post-Docs
Jean Quilbeuf, UBS Post-Doc grant, from December 2015 to December 2016

PhD students
Delphine Beaulaton, MESR grant, since October 2015
Raounak Benabidallah, ARED grant, since October 2016
Youcef Bouziane, Cotutelle grant, since September 2014
Everton Cavalcante, CsF-CNPq grant, defended his thesis on June 2016
Imane Cherfa, Cotutelle grant, since September 2014
Marcelo Gonçalves, CsF-CNPq grant, defended his thesis on December 2016
Valdemar Graciano Neto, CsF-CNPq grant, since November 2015
Milena Guessi, FAPESP grant, since September 2014
Soraya Mesli-Kesraoui, CIFRE grant with SEGULA, since November 2013
Franck Petitdemange, MESR grant, since September 2014
Lina Rodriguez, FAPESP grant, since December 2015
Mihai Enescu, ARED grant with INRIA, since October 2016 (co-supervision with TAMIS team)
Sébastien Martinez, ARED grant with Télécom Bretagne, defended his thesis on March 2016 (co-supervision with PASS team)

2 Overall Objectives

2.1 Overview

The ArchWare Research Team addresses the scientific and technological challenges raised by architecting complex software-intensive systems. Beyond large-scale distributed systems, it addresses an emergent class of evolving software-intensive systems that is increasingly shaping the future of our software-reliant world, the so-called Systems-of-Systems (SoS).

Since the dawn of computing, the complexity of software and the complexity of systems reliant on software have grown at a staggering rate. In particular, software-intensive systems have been rapidly evolved from being stand-alone systems in the past, to be part of networked systems in the present, to increasingly become systems-of-systems in the coming future.

De facto, systems have been independently developed, operated, managed, and evolved. Progressively, networks made communication and coordination possible among these autonomous systems, yielding a new kind of complex system, i.e. a system that is itself composed of systems. These systems-of-systems are evolutionary developed from systems to achieve missions not possible by each constituent system alone.

Different aspects of our lives and livelihoods have become overly dependent on some sort of software-intensive system-of-systems. This is the case of systems-of-systems found in different areas as diverse as aeronautics, automotive, energy, healthcare, manufacturing, and transportation; and applications that addresses societal needs as e.g. in environmental monitoring, distributed energy grids, emergency coordination, global traffic control, and smart cities.

Moreover, emergent platforms such as the Internet of Things and the Internet of Everything and emergent classes of systems-of-systems such as Cyber-Physical Systems are accelerating the need of constructing rigorous foundations, languages, and tools for supporting the architecture and engineering of resilient systems-of-systems.

Complexity is intrinsically associated to systems-of-systems by its very nature that implies emergent behavior: in systems-of-systems, missions are achieved through emergent behavior drawn from the interaction among constituent systems. Hence, complexity poses the need for separation of concerns between architecture and engineering: (i) architecture focuses on reasoning about interactions of parts and their emergent properties; (ii) engineering focuses on designing and constructing such parts and integrating them as architected.

Definitely, Software Architecture forms the backbone for taming the complexity of critical software-intensive systems, in particular in the case of systems-of-systems, where architecture descriptions provide the framework for designing, constructing, and dynamically evolving such complex systems, in particular when they operate in unpredictable open-world environments.

Therefore, the endeavor of constructing critical systems evolved from engineering complicated systems in the last century, to architecting critical SoSs in this century. Critical SoSs, by their very nature, have intrinsic properties that are hard to address.

Furthermore, the upcoming generation of critical SoSs will operate in environments that are open in the sense of that they are only partially known at design-time. These open-world critical systems-of-systems, in opposite to current closed-world systems, will run on pervasive devices and networks providing services that are dynamically discovered and used to deliver more complex services, which themselves can be part of yet more complex services and so on.

Furthermore, they will often operate in unpredictable environments.

Besides, in SoSs, architectures are designed to fulfill specified missions. Indeed, an important concern in the design of SoSs is the systematic modeling of both global and individual missions, as well as all relevant mission-related information. Missions play a key role in the SoS context since they define required capabilities of constituent systems and the interactions among these systems that lead to emergent behaviors towards the accomplishment of the global mission of the SoS.

Definitely, the unique characteristics of SoS raise a grand research challenge for the future of software-reliant systems in our industry and society due to its simultaneous intrinsic features, which are:

1. *Operational independence:* the participating systems not only can operate independently, they do operate independently. Hence, the challenge is to architect and construct SoS in a way that enables its operations (acting to fulfill its own mission) without violating the independence of its constituent systems that are autonomous, acting to fulfill their own missions.

2. *Managerial independence:* the participating systems are managed independently, and may decide to evolve in ways that were not foreseen when they were originally composed. Hence, the challenge is to architect and construct a SoS in a way that it is able to evolve itself to cope with independent decisions taken by the constituent systems and hence be able to continually fulfill its own mission.

3. *Distribution of constituent systems:* the participating systems are physically decoupled. Hence, the challenge is to architect and construct the SoS in a way that matches the loose-coupled nature of these systems.

4. *Evolutionary development:* as a consequence of the independence of the constituent systems, a SoS as a whole may evolve over time to respond to changing characteristics of its environment, constituent systems or of its own mission. Hence, the challenge is to architect and construct SoS in a way that it is able to evolve itself to cope with these three kinds of evolution.

5. *Emergent behaviors:* from the collaboration of the participating systems may emerge new behaviors. Furthermore, these behaviors may be ephemeral because the systems composing the SoS evolve independently, which may impact the availability of these behaviors. Hence, the challenge is to architect and construct a SoS in a way that emergent behaviors and their subsequent evolution can be discovered and controlled.

In the case of an open-world environment, one can add the following characteristics:

1. *Unpredictable environment:* the environment in which the open-world SoS operates is only partially known at design-time, i.e. it is too unpredictable to be summarized within a fixed set of specifications, and thereby there will inevitably be novel situations to deal with at run-time. Hence, the challenge is to architect and construct such a system in a way that it can dynamically accommodate to new situations while acting to fulfill its own mission.

2. *Unpredictable constituents:* the participating systems are only partially known at design-time. Hence, the challenge is to architect and construct an open-world SoS in a way that constituent systems are dynamically discovered, composed, operated, and evolved in a continuous way at run-time, in particular for achieving its own mission.

3. *Long-lasting:* as an open-world SoS is by nature a long-lasting system, re-architecting must be carried out dynamically. Hence, the challenge is to evolutionarily re-architects and evolves its construction without interrupting it.

The importance of developing novel theories and technologies for architecting and engineering SoSs is highlighted in several roadmaps targeting year 2020 and beyond.

In France, it is explicitly targeted in the report prepared by the French Ministry of Economy as one of the key technologies for the period 2015-2025 (étude prospective sur les technologies clés 2015-2025, Direction Générale de la Compétitivité, de l'Industrie et des Services du Ministére de l'Economie).

In Europe, SoSs are explicitly targeted in the studies developed by the initiative of the European Commission, i.e. Directions in Systems-of-Systems Engineering, and different Networks of Excellence (e.g. HiPEAC) and European Technological Platforms (e.g. ARTEMIS, NESSI) for preparing the Horizon 2020 European Framework Programme. In 2014, two roadmaps for systems-of-systems having been proposed, supported by the European Commission, issued from the CSAs ROAD2SoS (Development of Strategic Research and Engineering Roadmaps in Systems-of-Systems) and T-Area-SoS (Trans-Atlantic Research and Education Agenda in Systems-of-Systems).

All these key actions and the roadmaps for 2015-2020-2025 show the importance of progressing from the current situation, where SoSs are basically developed in ad-hoc way, to a scientific approach providing rigorous theories and technologies for mastering the complexity of software-intensive systems-of-systems.

Overall, the long-term research challenge raised by SoSs calls for a novel paradigm and novel trustful approaches for architecting, analyzing, constructing, and assuring the continuous correctness of systems-of-systems, often deployed in unpredictable environments, taking into account all together their intrinsic characteristics.

Actually, in the literature, SoSs are classified according to four categories:

1. *Directed SoS:* a SoS that is centrally managed and which constituent systems have been especially developed or acquired to fit specific purposes in the SoS - they operate under tight subordination;

2. *Acknowledged SoS:* a SoS that is centrally managed and that operates under loose subordination - the constituent systems retain their operational independence;

3. *Collaborative SoS:* a SoS in which there is no central management and constituent systems voluntarily agree to fulfill central purposes;

4. *Virtual SoS:* a SoS in which there is no central authority or centrally agreed purpose.

Regarding the state-of-the-art, software-intensive system-of-systems is an emergent domain in the research community. The systematic mapping of the literature shows that 75% of the publications related to the architecture of systems-of-systems have been published in the last 5 years and 90% in the last 10 years. Furthermore, most of these publications raise open-issues after having experimented existing approaches for architecting systems-of-systems.

Our last systematic literature review searching all key bibliographic databases relevant to computer science, software and systems engineering, looking for publications from academia and experience reports from industry shows that, today, proposed approaches only support the architecture and construction of SoS matching the core characteristics, basically directed and acknowledged SoSs limited to non-critical systems. Therefore, achieving the targeted breakthrough will be a major advance in the state-of-the-art by delivering a conceptual, theoretical, and technological foundation for architecting and constructing the new generation of critical SoSs, i.e. collaborative and virtual SoSs. For addressing the scientific challenge raised for architecting SoS, the targeted breakthrough for the ArchWare Research Team is to conceive sound foundations and a novel holistic approach for architecting open-world critical software-intensive systems-of-systems, encompassing:

1. Architectural abstractions for formulating the architecture and re-architecture of SoS;
2. Formalism and underlying computational model to rigorously specify the architecture and re-architecture of SoS;
3. Mechanisms to construct, manage, and evolve SoSs driven by architecture descriptions, while resiliently enforcing their correctness, effectiveness, and efficiency;
4. Concepts and formalisms for specifying and operating SoS missions and generating abstract and concrete SoS architectures.

The research approach we adopt in the ArchWare Research Team for developing the expected breakthrough is based on well-principled design decisions:

1. To conceive architecture description, analysis, and evolution languages based on suitable SoS architectural abstractions;
2. To formally ground these SoS-specific architecture languages on well-established concurrent constraint process calculi and associated logics;
3. To conceptually and technologically ground the construction and management of SoSs on architecture descriptions defined by executable models;
4. To derive/generate abstract/concrete architectural descriptions from well-defined mission specifications.

2.2 Key Issues

As stated, an SoS can be perceived as a composition of systems in which its constituents, i.e. themselves systems, are separately discovered, selected, and composed possibly at run-time to form a more complex system, the SoS. Hence, four points are at the core of our approach:

- SoS architectures as dynamic compositions of systems: An SoS depends on composed systems and their interactions to undertake capabilities. Composition is thereby more challenging in SoS as systems being combined are independent. This challenge beyond the state-of-the-art will be overcome in our approach by the development of SoS-specific composition mechanisms that are explicit, formally defined, and operate on independent constituent systems.

- Analysis of SoS architectures: Formal architecture descriptions aim to support automated analysis with a view in particular to evaluating and predicting non-functional qualities of the modeled SoS. In our approach, we make a significant progress beyond the state-of-the-art of SoS analysis by developing techniques and tools for the architecture-centric analysis of SoS. It involves the verification of different sorts of properties and interleaving of these properties, including structural (e.g. connectivity, topology), behavioral (e.g. safety, liveness, fairness), and quality properties (e.g. agility). This challenge beyond the state-of-the-art will be overcome in our approach by the development of different complementary analysis techniques combining simulation, model checking, and testing. The aim is to enable analysis and validation of a SoS architecture anytime along the whole SoS life-cycle by means of automated verification.

- Architecture-driven construction of SoSs: SoS architecture will drive the initial construction and subsequent evolutions of a SoS. This challenge beyond the state-of-the-art will be overcome in our approach by automatically generating concrete SoS architectures from abstract SoS architectural descriptions applied to a set of possible concrete constituent systems. Our approach relies on techniques for constraint satisfaction, where abstract SoS architectures are concretized to "meet in the middle" with concrete selection and integration of discovered systems, consequently generating concrete SoS architectures.

- SoS architectures are designed to fulfill missions: Mission specifications are the starting point for designing SoS architectures and are used as a basis of their whole evolutionary development process. In a mission-based approach for designing software-intensive SoSs, the concretization of the mission specification is given by the derivation/generation of an SoS architecture description.

Keywords: Software Architecture, Architecture Description, Architecture Analysis, Mission Specification, Software-intensive Systems-of-Systems.

3 New Results

3.1 The SoS Architecture Description Language (SosADL)

Keywords: Architecture Description Language (ADL), Systems-of-Systems (SoS).

Participants: Flavio Oquendo, Jérémy Buisson, Elena Leroux, Gersan Moguérou.

The architecture provides the right abstraction level to address the complexity of Software-intensive Systems-of-Systems (SoSs). The research challenges raised by SoSs are fundamentally architectural: they are about how to organize the interactions among the constituent systems to enable the emergence of SoS-wide behaviors and properties derived from local behaviors and properties by acting only on their connections, without being able to act in the constituent systems themselves.

Formal architecture descriptions provide the framework for the design, construction, and dynamic evolution of SoSs.

From the architectural perspective, in single systems, the controlled characteristics of components under the authority of the system architect and the stable notion of connectors linking these components, mostly decided at design-time, is very different from the uncontrolled nature of constituent systems (the SoS architect has no or very limited authority on systems) and the role of connection among systems (in an SoS, connections among constituents are the main architectural elements for enabling emergent behavior to make possible to achieve the mission of an SoS).

The nature of systems architectures (in the sense of architectures of single systems) and systems-of-systems are very different:

- Systems architectures are described by extension. In the opposite, SoS architectures are described by intension.
- Systems architectures are described at design-time for developing the system based on design-time components. In the opposite, SoS architectures are defined at run-time for developing the SoS based on discovered constituents.
- Systems architectures often evolves offline. In the opposite, SoS architectures always evolves online.

We have continued the development of an Architecture Description Language (ADL) specially designed for specifying the architecture of Software-intensive Systems-of-Systems (SoS). It provides a formal ADL, based on a novel concurrent constraint process calculus, coping with the challenging requirements of SoSs. Architecture descriptions are essential artifacts for (i) modeling systems-of-systems, and (ii) mastering the complexity of SoS by supporting reasoning about properties. In SosADL, the main constructs enable: (i) the specification of constituent systems, (ii) the specification of mediators among constituent systems, (iii) the specification of coalitions of mediated constituent systems.

SoS are constituted by systems. A constituent system of an SoS has its own mission, is operationally independent, is managerially independent, and may independently evolve. A constituent system interacts with its environment via gates. A gate provides an interface between a system and its local environment.

Constituent systems of an SoS are specified by system abstractions via gates, behavior and their assumed/guaranteed properties. Assumptions are assertions about the environment in which the system is placed and that are assumed through the specified gate. Guarantees are assertions derived from the assumptions and the behavior. Behavior satisfies gate assumptions (including protocols) of all gates.

Mediators mediate the constituent systems of an SoS. A mediator has its own purpose and, in the opposite of constituent systems, is operationally dependent of the SoS, is managerially dependent of the SoS, and evolves under control of the SoS.

Mediators among constituent systems of an SoS are specified by mediator abstractions. The SoS has total control on mediators. It creates, evolves or destroys mediators at runtime. Mediators are only known by the SoS. They enable communication, coordination, cooperation, and collaboration.

Coalitions of mediated constituent systems form SoSs. A coalition has its own purpose, may be dynamically formed to fulfill a mission through created emergent behaviors, controls its mediators

System-of-Systems are specified by SoS abstractions. The SoS is abstractly defined in terms of coalition abstractions. SoS are concretized and evolve dynamically at runtime. Laws define the policies for SoS operation and evolution. In SoSs, missions are achieved through the emergent behavior of coalitions.

3.2 A Novel Pi-Calculus for Systems-of-Systems

Keywords: Pi-Calculus, Process Algebra, Concurrent Constraints, System-of-Systems.

Participants: Flavio Oquendo.

In the case of Architecture Description Languages (ADLs) for describing the architecture of Software-intensive Systems, process calculi have shown to constitute a suitable mathematical foundation for modeling and analyzing system architectures.

ADLs based on process calculi have formalized: (i) components as defined processes in a specific process calculus, e.g. Darwin ADL in FSP, Wright ADL in CSP, and π-ADL in π-Calculus; (ii) connectors as structured bindings defined by explicit channels between communicating processes using a design-time binding mechanism, e.g. shared actions in Darwin/FSP, attachments in Wright/CSP, extruded channel names in π-ADL/π-Calculus.

The communication bindings in ADLs for the description of system architectures is: (i) decided at design-time, (ii) extensionally defined, (iii) unconstrained by local contexts, (iv) unmediated. By the nature of system architectures, the different process calculi capture the needs for the formalization of such ADLs.

However, none of the existing process calculi provides a suitable basis for modeling and analyzing the architecture of SoSs. Needs related to the description of SoS architectures are to represent: (i) systems as local defined processes, (ii) mediation between communicating processes using inferred bindings (i.e. actual bindings are decided at run-time by the SoS).

In the case of SoSs, the binding between channels must be: (i) decided at run-time, (ii) intentionally defined, (iii) constrained by local contexts, and (iv) mediated.

Precedently, we have evaluated the π-calculus (that subsumes other process calculi such as CSP, FSP, and CCS used as the basis of ADL formalization), in its original form and in the enhanced forms that have been developed along the years regarding the architecture description needs for SoS (binding needs to be decided at run-time, constrained by local contexts, intentional, and mediated). The conclusion of our evaluation was that binding in: (i) the basic π-Calculus is endogenous, unconstrained, extensional, and unmediated; (ii) the Fusion Calculus is exogenous, unconstrained, extensional, and unmediated; (iii) the explicit fusion π-F Calculus is exogenous, constrained, extensional, and unmediated; (iv) the concurrent constraints CC-π Calculus is exogenous, constrained, extensional, and unmediated; (v) the Attributed π-calculus is exogenous, constrained, extensional, and unmediated.

As none of them copes with the needs for SoS architecture descriptions, we have defined a novel process calculus for providing a suitable formal foundation of an ADL for SoS.

The design decisions underlying this novel π-Calculus, named π-Calculus for SoS are: (i) binding: binding between channels are designed to be exogenous, constrained by local contexts, intentional, and mediated; (ii) expressiveness: it is designed to be computationally complete (Turing completeness); (iii) style: it is designed to be architecture specification-oriented with support for recursion instead of replication; (iv) typing: it is designed to be strongly typed.

The approach for the design of the π-Calculus for SoS is to generalize the π-calculus with mediated constraints (mediation is achieved by constraining interactions), encompassing fusions, explicit fusions, concurrent constraints, and subsuming the basic π-calculus. The π-Calculus for SoS is parameterized by a call-by-value data expression language providing expressions to compute values and to impose constraints on interactions.

The SosADL is based on this novel calculus, the π-Calculus for SoS. Its formal semantics has been defined.

3.3 The SosADL Type System

Keywords: Type System, Architecture Description Language (ADL), Systems-of-Systems (SoS).

Participants: Jérémy Buisson, Gersan Moguérou, Flavio Oquendo.

Among the objectives of SosADL, we aim at designing a set of techniques and tools for the analysis and verification of SoS architectures. As part of this goal, we have initiated the definition of a type system. We follow the common approach, organizing the type system based on the structure of the language, as defined by the underlying meta-model. We implement a type checking and inference module in the SosADL toolchain.

In addition to the development itself, our focus in this work is to find out how all the artifacts (meta-model of the language, type system specification, and type checking and inference module) can more easily be kept synchronized in the context of a language whose definition will evolve with experimentation.

Our aim is therefore to bridge between semi-formal model-driven engineering and formal specification in this particular context. Our effort is currently targeted at two specific issues.

First, we aim at ensuring that the non-formal type checking and inference module conforms to the formally-defined type system. To do so, we follow an approach inspired by proof-carrying code: the type checking and inference module generates well-typedness proofs. Second, we aim at synchronizing the type system with the evolving abstract syntax of the language. To do so, we propose to use model transformation in order to produce some of the formal definitions from the meta-model of the language.

This work provides us a realistic case study about bridging between formal specification and model-driven infrastructure. It involves several transformations in both directions (but no bidirectional transformation), laying at various levels of abstraction (meta-meta-model, meta-model, and model). All the transformations must be consistent across all the levels such that, for instance, the resulting model at a given level is interoperable with the source level of another transformation at the level above. We intend to futher investigate how such a collection of consistent transformations can be built more easily, in order to lower the development burden.

3.4 SoS Analysis on SosADL: from ioSTS to NTA for SoS Verification with Uppaal

Keywords: Formal Analysis, Model Checking, Model Transformation, Software Architecture, Systems-of-Systems (SoS).

Participants: Elena Leroux, Gersan Moguérou, Flavio Oquendo.

The purpose of this research is to verify a set of functional properties for a given SoS architecture. The SoS architecture and its functional properties are expressed in SosADL. The semantics of SoS architecture descriptions are expressed using a labeled transition systems formalism, called input-output Symbolic Transition System (ioSTS). To allow the verification of properties of SoS architecture descriptions using the technique of model checking, we will use the Uppaal tool which takes a Network of Timed Automata (NTA) as an input. The goal is to automate the translation of the SosADL semantics expressed in ioSTS to the input format of Uppaal in terms of NTA. Note that in order to verify functional properties specified in SosADL, they also need to be translated into the Uppaal format. The properties which can be checked by Uppaal have to be expressed in a subset of computation tree logic (CTL).

We have defined the mappings between the ioSTS and NTA representations focusing on the specification of constituents and their composition in coalitions of SosADL.

3.5 Architectural Description by Constraints for SosADL

Keywords: Constraint Satisfaction Problem (CSP), Architectural Description, Systems-of-Systems (SoS).

Participants: Milena Guessi, Elisa Nakagawa, Flavio Oquendo.

Systems-of-Systems (SoSs) are evolutionary developed from independent systems to achieve missions through emergent behavior. As concrete systems that will actually participate in an SoS are, in general, not known at design-time, the dynamic establishment of new coalitions among participants of an SoSs must be supported.

While a declarative, abstract description of an SoS offers important guidelines for attaching constituents at runtime, efficient means for automatically realizing concrete architecture configurations that comply with such an abstract description are needed.

Aiming to provide such means, we developed an approach for representing the architectural configuration of an SoS in terms of a Constraint Satisfaction Problem (CSP).

We demonstrated the feasibility of such approach using a Flood Monitoring SoS (FMSoS), which is composed of heterogeneous types of sensors (embedded in distributed sensor motes) and gateways. Each sensor mote operates in a way that is independent of other sensor motes, as they belong to different city councils and have different missions, e.g. pollution control or water supply. Each sensor mote also has its own management strategy for transmission vs. energy consumption and will act under the authority of the different city councils. New sensor motes may be installed by the different councils as well as existing ones may be changed or uninstalled without any control from the SoS.

The sensor motes together, with the gateway, will make emerge the behavior of flood detection from the individual capabilities of the geographically distributed sensor motes to measure and re-transmit data and the gateway to analyze measured data. Given an SosADL description of the FMSoS, which specifies abstract types of sensor motes and gateways that can participate in the coalition, we are able to create corresponding Alloy models for five different runtime scenarios. Moreover, we were able to successfully determine the feasibility of such scenarios.

To support the transformation between SosADL and Alloy models for the SoS architectural configuration, we have defined a mapping between SosADL and the Alloy formal language. Reverse mappings are under development for transforming the concrete architecture that is realized by the Alloy Analyzer back to SosADL. Furthermore, an experiment is being carried out to assess the reliability and performance of this approach.

3.6 Statistical Model Checking of Dynamic Software Architectures

Keywords: Dynamic Software architecture, π-ADL, Statistical Model Checking.

Participants: Everton Cavalcante, Louis-Marie Traonouez (Inria Rennes), Flavio Oquendo, Axel Legay (Inria Rennes), Thais Batista (UFRN), Jean Quilbeuf.

Dynamic software architectures are those that encompass evolution rules for a software system and its elements during runtime. Their relevance is due to the fact that dynamism is an important concern for contemporary systems, which often operate on environments subjected to change.

In a dynamic software architecture, constituent elements may be created, interconnected or removed, or may even undergo a complete rearrangement at runtime, ideally with minimal or no disruption. One of the major challenges in the design of software-intensive systems consists in verifying the correctness of their software architectures, i.e., if the envisioned architecture is able to fully realize the established requirements.

One of the challenges for the verification of such properties is that existing approaches, such as model checking do not support unbounded creation of processes. Even if dynamic creation of processes were supported, existing approaches would suffer for the state space explosion problem.

Our approach for tackling these challenges relies on statistical model checking (SMC). A statistical model checker basically consists of a simulator for running the system under verification, a model checker for verifying properties, and a statistical analyzer responsible for calculating probabilities and performing statistical tests.

It receives three inputs: (i) an executable stochastic model of the target system; (ii) a formula expressing a bounded property to be verified, i.e., a property that can be decided over a finite execution of the model; and (iii) user-defined precision parameters determining the accuracy of the result. The checker performs several executions of the model. Since the model is stochastic, some executions might satisfy the formula and some other might not. Therefore, there is a value of probability that an execution of the model satisfies the formula. The checker returns an approximation of the value of probability, that complies with the accuracy parameters.

In order to express the formula to be verified, we need a logic that is able to reason about architectural properties. The particular nature of dynamic software architectures is that architectural elements (components and connectors) may appear, disappear, be connected or be disconnected at runtime. Therefore, expressing behavioral and structural properties regarding a dynamic software architecture needs to take into account architectural elements that are dynamically created and removed, i.e., they may exist at a given instant in time and no longer exist at another.

Finally, we need to be able to describe the dynamic software architecture. To this end, we rely on π-ADL, an extension of the π-calculus developped in the ArchWare team to describe such architectures. Simulation of π-ADL specifications is performed by generating go code.

In this research, we achieved the following results. First we defined the DynBLTL logic for expressing properties about dynamic architectures. Secondly, we developed a complete tool-chain for modeling and verifying dynamic software architecture using statistical model checking. We relied on the Plasma statistical model checker that we extended with two plugins. A first plugin implements the DynBLTL logic, enabling us to specify properties in this logic. A second plugin handle the communication between plasma and the simulation of π-ADL. Finally, we extended the π-ADL to go code generation to (1) control the simulation through the Plasma plugin and (2) implement a stochastic behavior. The tools are available at `http://plasma4pi-adl.gforge.inria.fr/`.

3.7 Statistical Model Checking of Systems-of-Systems Described with SosADL

Keywords: Dynamic Software architecture, π-ADL, Statistical Model Checking.

Participants: Valdemar Vicente Graciano Neto, Louis-Marie Traonouez (Inria Rennes), Flavio Oquendo, Axel Legay (Inria Rennes), Thais Batista (UFRN), Jean Quilbeuf.

The purpose of this research is to allow formal verification of systems-of-systems (SoS) described using SosADL by extending the SosADL architecture framework with appropriate techniques and tools. Indeed, it is crucial to guarantee that the important functional properties of SoS are held during the whole live cycle of this SoS. The challenge of such a verification is the large state space of the SoS, whose architecture may evolve.

In order to achieve our purpose, we rely on statistical model checking. Indeed, statistical model checking solely relies on a stochastic simulation of the model. We are extending the formal model of a given SoS with probabilities. Then, we will modify the existing SosADL simulator, based on the DEVS model, in order to obtain a stochastic simulation, and monitor its execution. Finally, we will rely on the Plasma statistical model checker to enable verification of the SosADL models. To this end, we provide a new simulator plugin, that interfaces with the DEVS simulator. In order to express properties, we rely on the DynBLTL logic, already available for Plasma.

This work extends the work carried for statistical model checking of dynamic software architectures described with π-ADL in order to support statistical model checking of dynamic SoS architectures described with SosADL based on the π-Calculus for SoS.

3.8 Defining SoS Patterns for SoS Reconfiguration

Keywords: Dynamic Reconfiguration, Pattern, System-of-Systems.

Participants: Franck Petitdemange, Isabelle Borne, Jérémy Buisson.

Systems-of-systems (SoS) are a particular class of systems that dynamically compose their constituents to achieve a global goal. To accommodate this approach, we propose that the SoS architecture be described by architectural patterns to be instantiated at runtime. Based on the study of cases, the objective of this research is to explore a new approach in order to reason about reconfiguration with specific patterns. It has mainly addressed the state-of-the-art for identifying open issues and proposed approaches. The focus is on patterns for dynamic reconfiguration.

We proposed an approach to describe a particular kind of SoS and how we intend to manage reconfigurations in a SoS architecture. The goal of this approach is to assist reconfiguration with a pattern-based approach. Relying on patterns implemented in SoSs, a set of reconfiguration patterns is selected and applied while maintaining the SoS architecture consistency. Reconfiguration implementation should follow the reconfiguration pattern. Our final objective is to obtain a catalog of architectural patterns devoted to SoS, built as a system of patterns where each pattern will describe an architectural solution unit to a SoS reconfiguration problem. An ongoing case study is based on a real situation: a flood monitoring system deployed in the city of Sao Carlos.

3.9 Describing and Managing SoS Evolution

Keywords: Dynamic Evolution, System-of-Systems.

Participants: Hichem Mohamed Fendali (Univ. Badji Mokhtar Annaba, Algeria), Isabelle Borne, Djamel Meslati(Univ. Badji Mokhtar Annaba, Algeria).

Evolution has always been a major issue for systems of any type. The evolution of a system is a necessity imposed, both by the evolution of the support platforms and by the changing needs of the users. The major disadvantage of the management of evolution in classical systems is that it is considered a sporadic process that introduces changes when in reality evolution is a continuous process that has shaped the system since its creation. In systems-of-systems, evolution is perceived as a central concern. Its capture, description, management, control of its scope and the search for compromise that it imposes require the development of specific approaches and dedicated mechanisms that are both general and abstract in order to take account of the heterogeneity of the constituent systems, but also sufficiently explicit and precise to be exploitable in practice.

Work on the evolution of SoS raises many questions that needs to be answered. In a first step, it is necessary to identify the nature of the evolution in the SoS. Although the latter derives from the evolution of the constituents or the mission assigned to the SoS, it is important to limit oneself to the architectural aspect. In other words, the interest will be focused on a strong granularity that ignores the small changes affecting the constituents and which must be supported dynamically by these constituents and dynamically also by the SoS. A state of the art was produced based on change motivations and challenges of current research works that contribute to SoS evolution.

3.10 Automatic Refactoring of Component-Based Software by Detecting and Eliminating Bad Smells

Keywords: Refactoring, Component-Based Software.

Participants: Isabelle Borne, Hichem Mohamed Fendali (Univ. Badji Mokhtar Annaba, Algeria), Djamel Meslati (Univ. Badji Mokhtar Annaba, Algeria), Salim Kebir (Univ. Badji Mokhtar Annaba, Algeria).

Refactoring has been proposed as a de facto behavior-preserving mean to eliminate bad smells. However manually determining and performing useful refactorings is a though challenge because seemingly useful refactorings can improve some aspect of a software while making another aspect worse. Therefore it has been proposed to view object-oriented automated refactoring as a search-based technique. Nevertheless the review of the literature shows that automated refactoring of component-based software has not been investigated yet. Recently a catalogue of component-relevant bad smells has been proposed in the literature but there is a lack of component-relevant refactorings. The problem addessed is to automate refactoring of component-based software systems. This project proposes detection rules for component-relevant bad smells as well as a catalogue of component-relevant refactorings. We rely on these two elements to propose a search-based approach for automated refactoring of component-based software systems by detecting and eliminating bad smells. The approach was experimented on a medium-sized component-based software and we assess the efficiency and accuracy of our approach.

3.11 Meta-Process for Describing SoS Architectures

Keywords: Architectural Process Model, Architectural Decisions, System-of-Systems (SoS).

Participants: Marcelo Gonçalves, Flavio Oquendo, Elisa Nakagawa.

The development of SoS differs from monolithic systems in several issues, such as the dynamic contribution and impact of constituent systems, which are developed and managed by independent sources. Thereby, SoS software architectures have reached a threshold in which traditional software architecture approaches are no longer adequate. Moreover, there is a lack with respect to processes and methods to construct SoS considering the design, representation, implementation, and evolution of their architectures.

Despite the relevance and necessity of software-intensive SoS for diverse application domains, most of their software architectures have been still developed in an *ad hoc* manner. In general, there is a lack of structured processes for architecting SoS, hindering the adoption of SoS.

For addressing this issue, we defined a meta-process independent of specific implementation technologies or application domains, named "Meta-process for SoS Software Architectures" (SOAR). With SOAR, software architects, process managers, and other SoS stakeholders can have support to instantiate their own processes when constructing SoS software architectures.

SOAR has been the result from an analysis of the state of the art of SoS development in conjunction with lessons learned with collaborating experts and investigations in real-world development environments. It can be valuable for several application domains and with the maturation of new good practices as standard solutions for SoS, new architectural decisions can be further incorporated to SOAR, yielding new versions for more specific contexts. Moreover, SOAR was produced with OMG's Essence Language through a modularized perspective in which the main scopes of problem concerning the construction of SoS software architectures were dealt (i.e., software development, construction of software architectures, and construction of SoS software architectures). This modularization allows the best understanding and application of SOAR as a meta-process.

Relying on the OMG's Essence Language, we defined a kernel and three practices for the SOAR SoS architecture process model: (i) SOAR Kernel: General Approach for Architecting Acknowledged SoS; (ii) SOAR-A: Architectural Analysis on Acknowledged SoS; (iii) SOAR-S: A Practice for Architectural Synthesis on Acknowledged SoS; (iv) SOAR-E: A Practice for Architectural Evaluation on Acknowledged SoS. We also continued the work to evaluate the applicability of SOAR, through experimental studies.

3.12 Architectural Design of Service-oriented Robotic Systems

Keywords: Service Oriented Architecture, Robotic Systems, System-of-Systems, Architectural Decisions.

Participants: Lucas Oliveira, Flavio Oquendo, Elisa Nakagawa.

Robots have increasingly supported different areas of the society, making daily tasks easier and executing dangerous, complex activities. Due to this high demand, robotic systems used to control robots are becoming larger and more complex, which creates a great challenge to the development of this special type of software system. Over the last years, researchers have been adopting the Service-Oriented Architecture (SOA) architectural style as a solution to develop more reusable, flexible robotic systems. Several studies in the literature report the creation of Service-Oriented Robotic Systems (SORS), as well as new languages and environments for the development of such systems. Nevertheless, few attention has been paid to the development of SORS software architectures. Currently, most of software architectures are designed in an *ad hoc* manner, without a systematic approach of development, hampering the construction, maintenance, and reuse of robotic systems. The consideration of quality attributes since the software architecture design is a critical concern, as these systems are often used in safety-critical contexts.

To mitigate this issue, we first defined a process named ArchSORS (Architectural Design of Service-Oriented Robotic System), which aims at filling the gap between the systematic development of service-oriented systems and the current *ad hoc* approaches used to develop SORS. The ArchSORS process provides prescriptive guidance from the system specification to architecture evaluation. Following, we established a reference architecture to support the design of SORS software architectures developed using ArchSORS. The reference architecture, named RefSORS (Reference Architecture for Service-Oriented Robotic System), is based on different sources of information, such as: (i) SORS available in the literature identified by means of a systematic review, (ii) a taxonomy of services for developing SORS, established in conjunction of robotics specialists, (iii) reference models and reference architectures available for SOA, (iv) reference architectures for the robotics domain, and (v) control architectures of the robotics domain. RefSORS encompasses multiple architecture views described in high-level representations and the semi-formal description languages SoaML (Service-oriented architecture Modeling Language) and UML (Unified Modeling Language).

Results of a controlled experiment involving students engaged in the French robotics competition RobaAFIS already pointed out that ArchSORS can improve coupling, cohesion, and modularity of SORS software architectures, which can result in systems of higher quality. The same RobaAFIS project adopted in the experiment was also developed in the context of a case study that uses the RefSORS reference architecture to support the application of the ArchSORS process. The software architecture designed in this case study presented better results in the three metrics (coupling, cohesion, and modularity) if compared to those created in the experiment by using only the ArchSORS process. The robotic system designed during the case study was development, showing the relevance of the proposed solution in terms of process and reference architecture.

3.13 Architecture-based Code Generation in Go for Dynamic Software-intensive Systems

Keywords: Architecture description languages, π-ADL, Implementation, Go.

Participants: Everton Cavalcante, Flavio Oquendo, Thais Batista.

A recurrent problem of almost all Architecture Description Languages (ADLs) is the decoupling between architecture descriptions and their respective implementations. As software architectures are typically defined independently from implementation, ADLs are often disconnected from the runtime level, thus entailing mismatches and inconsistencies between architecture and implementation mainly as the architecture evolves. Even though a system is initially built to conform to its intended architecture, its implementation may become inconsistent with the original architecture over time. This problem becomes worse with the emergence of new generation programming languages because most ADLs do not capture the new features of this type of languages, which are intended to take advantage of the modern multicore and networked computer architectures. Therefore, the architectural representation and system implementation need to be continuously synchronized in order to avoid architectural erosion and drift.

In order to support the transition from architecture description to implementation, the π-ADL was integrated with the Go language (http://golang.org/), a new programming language recently developed at Google. Unlike most existing ADLs, π-ADL provides a formal language for describing dynamic software architectures by encompassing structural and behavioral viewpoints, as well as supporting their automated, rigorous analysis with respect to functional and non-functional properties. In turn, Go was chosen to serve as implementation language because it is an easy general-purpose language designed to address the construction of scalable distributed systems and to handle multicore and networked computer architectures, as required by new generation software systems. The integration of π-ADL with Go was mainly fostered by their common basis on the π-calculus process algebra and the straightforward relationship between elements of the languages, such as the use of connections in π-ADL and channels in Go as means of communication between concurrent processes.

The integration of π-ADL and Go resulted in comprehensive correspondences between the languages, which were used to develop a technical process aimed to automatically source code in Go from architecture descriptions in π-ADL. Furthermore, an Eclipse-based plug-in was built to assist software architects in the textual description of architectures using the π-ADL language and to generate source code in Go. Therefore, when describing a software architecture in π-ADL, if it is correct according to the syntactic and semantic rules of the language (verified by the textual editor), then the respective Go source code is generated with the automatic build capability provided by the π-ADL textual editor.

After having addressed the translation from π-ADL to Go for static architectures, we have addressed the issue of dynamic architectures.

3.14 Formal Verification and Generation for Reconfigurable Socio-technical Systems

Keywords: formal verification, requirements modelling, model checking.

Participants: Soraya Mesli, Pascal Berruet (Lab-STICC), Alain Bignon (SEGULA), Flavio Oquendo.

The design of socio-technical systems is an activity more and more complex because it involves several designers from very different technical backgrounds. This variety of profiles causes to comprehension difficulties of specifications, which reflects the high number of errors usually detected during the product testing stage.

To minimize theses errors in the earlier stages of designing, a model-driven engineering approach was proposed. This approach permits to each designer to concentrate in his heart craft. The bridge between the designers is realized by a succession of models transformations.

Most models were generated automatically. The approach cited above makes consistency between the generated models. But it does not allow formal verification of these models. Indeed, if the source model contains design errors due to misinterpretation of the specifications, automatic generation causes the propagation of these errors to the generated models.

The main goal of this work is to introduce in the earlier stages of the aforementioned approach a set of formal verification techniques to obtain secure systems with reduced cost and respecting the customer requirements.

Our proposed approach is divided in three steps. The goal of each step is expressed by a question. These tree questions are: (i) What we should verify?; (ii) How we should verify ?;(iii) Where we should verify?

What we should verify? The goal of this step is to determine the most important properties that should be verified. To get a list of all important properties, we have made four semi structured interview with different designers. These interviews permit us to establish an initial list of properties. This initial list will be completed by the state of art of common verified properties. At the end, we will obtain a complete list of properties that should be verified.

How we should verify? The purpose of this question is to choose among all the existing verifications methods, the more adequate of kind of property and our system. We have elaborated a protocol of systematic mapping in this field.

Where we should verify? The main of this step is to determine what type of property should be formally verified on which system model. The complete list will allow us to make this decision. System model should be translated into a mathematical model. We will use model checking to verify the property on the model.

Different component features (the control program, the supervision interface, the physical device) and the human tasks are modeled using timed automata. These timed automata are then checked by model checking, applying Uppaal), with a set of safety and usability properties written in CTL. Our approach was applied to an industrial case study. The results showed that the use of formal techniques enables to successfully detect control program and supervision interface design errors.

3.15 Method for Limiting the Introduction of Vulnerabilities in Software

Keywords: Source Code Vulnerabilities, Design, Input Validation.

Participants: Jean Quilbeuf, Delphine Beaulaton, Salah Sadou, Régis Fleurquin.

Cybersecurity deals with preventing unaccredited users of computer systems to obtain or modify confidential information while maintaining the availability of the system for legitimate users. Although security is more and more taken into account by designers, we still discover vulnerabilities in today's systems. There are multiple reasons for this lack of security. First, despite the progress of software engineering, designing and building large systems remains a challenging task. In that context, even functional correctness is hard to achieve. Security properties are even more challenging, since the designer needs to identify the potential attackers, and how they could harm the system. The latest is facilitated by existing methodologies such as the common criteria and the EBIOS method. However, these methods results in a set of security requirements that are then left to the designer, and few tools and methodologies exist to track if these requirements are still met by the implementation. At the implementation level, security is often reduced to a set of best practices. These best practices tend to prevent vulnerabilities that are frequently identified in databases of vulnerabilities, such as CVE.

Systems-of-systems are even more challenging from a security point of view, as the composition of two individually secure systems may create an emergent behavior that an attacker can exploit during an attack.

The goal of this research is to conceive a method for limiting the introduction of vulnerabilities in systems, starting from the design phase. To that end, we aim to provide the designer a way to formalize the security requirements, and check at every step that the security requirements are still met, until the implementation. Our approach consists in adding a new model representing the needed security properties as well as the current state of the system. Such a view would allow the designer to reason about the security in the new system. Whenever the design of the system progresses, the security model should be updated. After an update, the security model would identify potential vulnerabilities introduced by the modification and enable the designer to correct it right away.

In order to obtain this security model, we focus on lower-level models. On one hand, we consider input validation, on the other hand we focus on information flows. We focus on input validation because the lack of input validation potentially causes the most often encountered vulnerabilities according to OWASP. Currently, some tools automatically detect the lack of input validation that causes a particular vulnerability. These tools are often limited to a particular language and a specific type of vulnerabilities. Our goal is to allow the designer to easily model the input validation mechanisms of the system. Based on this model, we plan to detect the inputs that are not sufficiently validated before entering a sensible part of the system, which may introduce vulnerabilities. The main challenge of this approach is to ensure that all elements involved in input validation have been detected. In particular, the inputs, the path followed by the inputs, the various types of validation needed by a system, and the points where these validations are necessary should all be identified.

Our approach is to provide a metamodel for representing the input validation of a system. This metamodel contains constraints that detect situations where an input is not validated enough before entering a sensible part of the system. Furthermore, we provide a methodology for building such a model from the source code. We plan to automatize the construction of this model as much as possible, both from existing code and from a higher-level model of security of the system. Another direction of our work focuses on controlling the flow of information inside a system. Indeed, even if access-based control prevents unauthorized users to access some part of the system, some confidential information might still be obtained by exploring the parts they have access to. Various tools allow specifying and verifying information flow at the source code level, which make is hard for the developer to have a global view of the security policy.

Our goal is to provide a view for specifying information flow properties at the system level. We also plan to provide some tools to help the developer add the information flow annotation to the source code.

3.16 Towards Automatic Detection of Vulnerabilities in Software-intensive Systems

Keywords: Source Code Vulnerabilities, Design, Input Validation.

Participants: Delphine Beaulaton, Jean Quilbeuf, Salah Sadou, Régis Fleurquin.

The security of a system is its ability to maintain the confidentiality, integrity and availability of its assets against potential attacks. In the frame of software-intensive systems, the assets are pieces of information. Confidentiality states that only authorized users of the system may access a given information. For instance, only the holder of a bank account can access its balance. Integrity states that only authorized users can modify a particular piece of information, i.e. only the holder of an account can order transfers from that account. Availability means that the information should be available at all times to authorized users. A security policy specifies the confidentiality, integrity and availability properties expected for each asset of a system.

Some systems are specifically designed to be secure such as banking applications or military communication systems. In that context, "secure" means "reasonably secure for their intended use". A classical approach for building complex systems is to interconnect simpler systems. Such an approach is not guaranteed to build a secure system, even if the simpler systems are secure themselves. Indeed the interconnection of several systems may introduce new vulnerabilities. A vulnerability can be seen as an unintended data or execution path in the system that allows an attacker to access, modify or make unavailable some assets. The Common Vulnerabilities and Exposures is a database that lists known vulnerabilities. Note that a system with vulnerabilities is possibly functionally correct, but unable to enforce its security policy.

Detecting vulnerabilities is challenging, in particular when they arise from the composition of several systems. Several methods exist for detecting specific vulnerabilities, however all possible vulnerabilities are not covered. Therefore our goal in this work is to provide a methodology allowing the detection of weaknesses, indicating to the designer which parts of the system are possibly vulnerable. Thanks to this more abstract level (weakness vs. vulnerablity), we hope to discover more vulnerabilies than existing approaches.

In order to tackle this problem, we propose to build a model of the system that helps detecting its possible weaknesses. Such a model should state the policy security of the system as well as its assets. Furthermore the model should include a description of the system, either generated from existing code, or other documents, that enables to state vulnerabilities as structural properties of that model. In order to build this model, we rely on Common Weakness Enumeration (CWE), a database that classifies weaknesses. We aim to be able to model systems where each weakness class from CWE is transformed into a property on the model.

3.17 Definition of an SoS Mission Description Language

Keywords: Mission Description Language, System-of-Systems, Goal-orientation.

Participants: Eduardo Silva, Everton Cavalcante, Flavio Oquendo, Thais Batista.

A System-of-Systems (SoS) is architected as a composition of constituent systems which are independent and operatable, and which are composed together for a period of time to achieve a specified mission. The SoS mission is thereby an essential statement that can guide the whole SoS development process. Through the so-called mission specification, it is possible to identify required capabilities for the constituent system, operations, connections, emergent behavior, among other elements that characterize an SoS.

In an SoS, each constituent system has its own mission that it must achieve independently, while concurrently collaborating with other constituent systems for the achievement of a common SoS mission it participates in. An important challenge for the architectural design of an SoS is to systematically specify the SoS mission and all mission-related information.

In particular, a mission is an operational goal that plays a key role in the SoS architectural design, since it guides the whole development process by defining the required capabilities of the constituent systems, since constituent systems must implement specific capabilities to contribute to the mission achievement. Likewise, the specification of the desired and/or necessary emergent behaviors is also related to the missions.

Although the importance of missions for the SoS domain, the literature provides few proposals that focus on SoS missions and none of them encompass a conceptual model for representing missions or a language to specify SoS missions. For addressing this gap, we have defined mKAOS as a language for SoS mission description that allows designers: (i) to describe global and individual missions, and (ii) to relate these concepts to various aspects of the system, such as constituent systems, emergent behaviors, and system capabilities. It promotes a clear separation of the mission description from the SoS architecture description. It is based on and extends KAOS, a goal-oriented requirement specification language.

3.18 Meta-model for an SoS Mission Description Language

Keywords: Mission Description Language, System-of-Systems, Meta-Models, DSL.

Participants: Imane Cherfa, Salah Sadou, Régis Fleurquin, Nicolas Belloir.

One of the key challenges in the field of Systems-of-Systems (SoS) is to ensure that an SoS behavior is maintained despite the constant changes in the SoS architecture. This obviously implies a constant re-architecture of the SoS to adapt to these changes. This alters the status of the SoS architecture model that becomes not stable enough to be the 'knowledge' on which we can control and infer future changes. Indeed, the concept of mission is an important part of this knowledge. The mission of an SoS remains the same whatever is its architecture. Thus, we study 4 languages (Kaos, Tropos, BPMN and BPEL), often used in the field of Requirement Engineering (Kaos, Tropos) and Process Modeling (BPMN, BPEL), to extract useful concepts for the definition of missions. These languages are potential candidates for the modeling of all or part of missions. Based on this study and some specific characteristics of SoS, we identified a list of 10 important concepts to describe a mission. We then showed that none of the four languages offers all of the concepts needed for supporting modeling.

Based on this study, we defined a new mission modeling language from scratch as a domain specific language. Its meta-model, namelly MDL, was defined. This work complements the research carried out in the team, which extended Kaos for defining a novel mission specification language, named mKAOS.

3.19 Situation/Reaction as a Paradigm for Defining SoS Missions

Keywords: Mission Description Language, System-of-Systems, Situation/Reaction.

Participants: Rymel Benabidallah (USTHB), Salah Sadou, Mohamed Ahmed Nacer (USTHB), Régis Fleurquin.

The collaboration of constituent systems in a System-of-Systems (SoS) is intended to achieve a global mission. Each constituent system has an independent behaviour for accomplishing its own individual mission, and may participate in the global mission of the SoS.

This research explores a new approach to architect SoS. When trying to describe the emergent behavior of SoS, it is necessary to fix the concepts participating in the construction of the SoS and its mission. We developed a metamodel to describe the environment in which the SoS operates to reach its mission. The concretization of our approach is achieved thanks to a combination of an expert system with an orchestration language. In one hand the expert system describes the environment, stakeholders and the events or changes that happen in the environment. In the other hand, reactions are represented as orchestration of activities.

3.20 Using Object-Oriented Metrics to Predict Component-Based Applications Defects

Keywords: Object-Oriented Metrics, Component-Based Development, Prediction Model.

Participants: Salma Hamza, Salah Sadou, Régis Fleurquin, Lucas Oliveira (USP).

Component-Based Development (CBD) has been claimed as a step toward quality in large software systems. However, unlike the traditional Object-Oriented (OO) development, CBD paradigm still lacks of internal metrics able to control and predict quality characteristics.

In this research, we showed that some well-known OO metrics can be used to build defect prediction models for component-based software systems. Thus, we applied consolidated OO code metrics on six large open-source systems developed using OSGi framework in order to create models able to predict defects. These prediction models were created using Principal Component Analysis and Multivariate Logistic Regression. Results of our study pointed out that the built models can predict 80% to 92% of frequently defective components with recall ranges between 89% and 98%, depending on the evaluated project. It is, therefore, possible to conclude that the considered OO metrics are able to successfully predict certain external quality properties in component-based systems.

4 Software

4.1 SosADE: The SoS Architect Studio for SosADL

Participants: Gersan Moguérou, Jérémy Buisson, Elena Leroux, Flavio Oquendo.

SosADE, the SosADL Architecture Development Environment, is a novel environment for description, analysis, simulation, and compilation/execution of SoS architectures. These SoS architectures are described using SosADL, an Architecture Description Language based on process algebra with concurrent constraints, and on a meta-model defining SoS concepts. Because constituents of an SoS are not known at design time, the language promotes a declarative approach of architecture families. At runtime, the SoS evolves within such a family depending on the discovery of concrete constituents.

4.2 SosADL2ioSTS: SosADE Support for Verification

Participants: Elena Leroux, Gersan Moguérou.

SosADL2ioSTS is a part of the SosADE SoS Architecture framework which purpose is to represent the behaviors of SoSs expressed using the SosADL language as ioSTS models in order to verify interesting and important functional properties of SoS by giving this model to different existing tools used for verification of software systems.

The development of a translator from ioSTS to Uppaal started. The SoSADL type system has been developed in Coq, and it will be used to develop the SosADL type-checker, which will produce Coq proofs.

4.3 SosADL2Alloy: Generating Concrete SoS Architectures based on SosADL

Participants: Milena Guessi, Flavio Oquendo.

SosADL2Alloy is an extension to the SosADL IDE, i.e. SosADE, for automatically transforming an abstract architecture in SosADL into an abstract architecture in Alloy. Given an SosADL project, the tool generates the corresponding Alloy files and calculates a predefined scope for executing the new models. The tool also generates a standard java class per SosADL architecture declaration which is responsible for calling one of the SAT solvers built in the Alloy Analyzer tool and saving the output in a separate file within the same generated folder. The user can manually change the execution scope and SAT solver in this class, if desired. The transformation of this output into a concrete architecture description expressed in SosADL is under development.

4.4 DynBLTL Plugin for Plasma Statistical Model Checker

Participants: Jean Quilbeuf, Louis-Marie Traonouez (Inria Rennes), Flavio Oquendo, Axel Legay (Inria Rennes).

The DynBLTL plugin for the Plasma statistical model checker enables the use of the DynBLTL logic. This logic was developped to express LTL properties about dynamic systems, where the set of components and their interconnections might evolve during the execution of the system.

4.5 Stochastic Simulator for Pi-ADL

Participants: Everton Cavalcante, Thais Batista, Jean Quilbeuf, Louis-Marie Traonouez (Inria Rennes), Flavio Oquendo, Axel Legay (Inria Rennes).

The goal of this tool is to enable the stochastic model checking for π-ADL. The tool consists of 3 components:

- a stochastic scheduler, written in Go,
- a π-ADL to Go code generator, that transform a π-ADL model into a Go program, that relies on the stochastic scheduler,
- a plugin for the Plasma statistical model checker, that interfaces with the stochastic scheduler and monitors the execution of the π-ADL model.

4.6 Coqcots & Pycots: Component Model and Framework for Supporting Dynamic Software Updating

Participants: Jérémy Buisson, Elena Leroux, Fabien Dagnat (Télécom Bretagne), Sébastien Martinez (Télécom Bretagne).

Coqcots & Pycots is a component model and framework under development on the gforge Inria as a collaborative work with the PASS team. Coqcots is a Coq model that allows the reconfiguration developer to formally specify, program then verify reconfiguration scripts. Pycots is the corresponding framework for the Python language.

The process is supported end-to-end. The architecture of a running Pycots application can be extracted using a reflexive level. This extraction builds a Coqcots model, a formal component model defined in Coq. Once the architecture is extracted, the developer simultaneously defines its reconfiguration and proves its correctness within Coq. Once proved, the reconfiguration script is extracted from the proof using Coq extraction facilities. Then this script is glued with dynamic software updating (DSU) code developed to support the updates of component behavior. Lastly, this code is uploaded to the Pycots running application to be executed. The resulting update therefore modify the application without stopping it.

Pycots relies on Pymoult, a library developed at Télécom Bretagne, which provides many DSU mechanisms in a single platform.

As part of the project, the extraction plugin of Coq has been extended in order to support Python output.

4.7 Pymoult: Prototyping Platform for Dynamic Software Updating

Participants: Sébastien Martinez (Télécom Bretagne), Fabien Dagnat (Télécom Bretagne), Jérémy Buisson.

Pymoult is a Pypy library providing a prototyping platform for Dynamic Software Updating (DSU). Many of the mechanisms from the literature has been reimplemented in Pymoult. The library then allows to recombine these mechanisms at developer' wish in order either to simulate other existing platforms or to experiment new combinations. Currently, more than 15 existing platforms can be simulated with Pymoult.

4.8 PiADL2Go: Supporting Dynamic Software Architectures from Pi-ADL to Go

Participants: Everton Cavalcante, Flavio Oquendo, Thais Batista (UFRN).

PiADL2Go is the result of the implementation of the π-ADL architectural language with the Go programming language. On the one hand, π-ADL provides a formal, theoretically well-founded language for describing dynamic software architectures by encompassing both structural and behavioral viewpoints, unlike most existing architecture description languages (ADLs). In turn, Go is an easy general-purpose language designed to address the construction of scalable distributed systems and handle multicore and networked computer architectures. In this perspective, the correspondences between the elements of these languages were defined and a process that defines how architecture descriptions in π-ADL automatically are translated to their respective source code implementations in Go was developped. The code generator within the π-ADL textual editor is implemented by using facilities provided by the Xtend programming language.

5 Contracts and Grants with Industry

5.1 Grants Involving Industry

Collaboration on architectural frameworks for business intelligence (CIFRE)
Participants: MGDIS.

Collaboration on verification of naval systems and systems-of-systems architectures (CIFRE)
Participants: SEGULA.

5.2 International Grants - Cooperative Projects

SASoS (Supporting Development of Software Architectures for Software-intensive Systems-of-Systems)

- Funding: FAPESP (Sao Paulo State Research Agency)

- Period: 2014 - 2016
- Partners: University of Sao Paulo - ICMC Research Institue (Brazil)
- Objective: To develop a framework for architecting Software-intensive Systems-of-Systems and applying results for architecting a flood monitoring SoS.

ArchIoT (Software Architecture for the Internet-of-Things)

- Funding: INES-CNPq (National Institute for Software Engineering - National Research Agency)
- Period: 2014 - 2016 (extended)
- Partners: UFRN - Federal University of Rio Grande do Norte (Brazil)
- Objective: To develop an ADL based on SysML for architecting applications to be deployed on the Internet-of-Things.

6 Other Grants and Activities

6.1 International Collaborations

- Flavio Oquendo:
  - USP - University of Sao Paulo - ICMC Research Institute, Sao Carlos, Brazil (Elisa Nakagawa): Architecting critical systems-of-embedded systems (PhDs in co-tutelle)
  - UFRN - Federal University of Rio Grande do Norte, Natal, Brazil (Thais Batista): Architecting dynamic software-intensive systems-of-systems (PhDs in co-tutelle)
- Salah Sadou:
  - University of Science and Technology of Houari Boumedienne, Alger, Algeria (Mohamed Ahmed Nacer): Product line architecture for systems-of-systems (PhD in co-direction)
  - University of Blida, Algeria: Mission description language (PhD in co-tutelle)
- Isabelle Borne:
  - LISCO, University Badji Mokhtar Annaba, Algeria (Djamel Meslati): Model driven engineering approach to design mobile agent applications (PhD in co-tutelle)
  - LISCO, University Badji Mokhtar Annaba, Algeria (Djamel Meslati): Automatic refactoring of component-based software by detecting and eliminating bad smells (PhD co-supervisor)
  - LISCO, University Badji Mokhtar Annaba, Algeria (Djamel Meslati): Evolution of Systems-of-Systems ((Phd co-supervisor)

6.2 National Collaborations

- Jérémy Buisson has a collaboration with Télécom Bretagne (PASS team), contributing to the supervision of the PhD student Sébastien Martinez
- Flavio Oquendo has a collaboration on systems-of-systems with Khalil Drira (LAAS-CNRS) and Axel Legay (INRIA)
- Salah Sadou has a collaboration on reuse of architectural constraints with Chouki Tibermancine and Christophe Dony (LIRMM)

6.3 Local Collaborations

- Salah Sadou leads a project on Cybersecurity with 6 Post-Docs funded by the Brittany council and the UBS. The Post-Docs are allocated in 5 laboratories of the UBS in different related disciplines.

7 Dissemination

7.1 Editorial Boards and Guest Editions

- Flavio Oquendo:
  - Springer Journal of Software Engineering Research and Development (Member of the Editorial Board)
  - International Journal of Artificial Intelligence and Applications for Smart Devices (Member of the Editorial Board)
  - Special Issue on Adaptive and Reconfigurable Service-oriented and Component-based Applications and Architectures of the Inderscience International Journal of Autonomous and Adaptive Communications Systems (Guest Editor)

7.2 General Chairs, Steering Committees

- Flavio Oquendo:
  - European Conference on Software Architecture - ECSA (Steering Committee Chair)
  - IEEE/IFIP Working International Conference on Software Architecture - WICSA (Steering Committee Member)
  - Conférence francophone sur les architectures logicielles - CAL (Steering Committee Member)
  - IEEE International Conference on Collaboration Technologies and Infrastructures - WETICE (Steering Committee Member)
  - ACM International Workshop on Software Engineering for Systems-of-Systems (technically co-sponsored by ACM SIGSOFT and ACM SIGPLAN) - SESOS (Steering Committee Chair)

  - Workshop on Distributed Development of Software, Ecosystems and Systems-of-Systems - WDES (Steering Committee Member)

- Salah Sadou:
  - CIEL: French Conference on Software Engineering, 2016 (Steering Committee Member)

7.3 Program Chairs, Tutorial Chairs, Special Session Chairs

- Flavio Oquendo:

  ACM International Workshop on Software Engineering for Systems-of-Systems (SESoS) at ACM/IEEE International Conference on Software Engineering (ICSE), Austin, Texas, USA, 2016 (Program Chair)

- Salah Sadou:
  - Advanced in Software Engineering Track of the ACS/IEEE International Conference on Computer Systems and Applications (AICCSA), Agadir, Maroc, 2016 (Program Chair)

7.4 Program Committees

- Isabelle Borne:
  - SESOS: ACM/IEEE ICSE International Workshop on Software Engineering for Systems-of-Systems, 2016
  - AROSA: IEEE WETICE Conference Track on Adaptive and Reconfigurable Service-oriented and Component-based Applications and Architectures, 2016
  - ASOCA: ICSOC Workshop on Adaptive Service-Oriented and Cloud Applications, 2016
- Jérémy Buisson:
  - ICCS: International Conference on Computational Science, 2016
  - C&ESAR: Computer & Electronics Security Applications Rendez-vous, 2016
- Régis Fleurquin
  - Advanced in Software Engineering Track of ACS/IEEE AICCSA, 2016
- Flavio Oquendo:
  - WICSA: IEEE/IFIP Working International Conference on Software Architecture, 2016
  - ECSA: European Conference on Software Architecture, 2016

  - QoSA: ACM International Conference on the Quality of Software Architectures, 2016
  - CBSE: International Symposium on Component Based Software Engineering, 2016
  - ICSEA: International Conference on Software Engineering Advances, 2016
  - I-ESA: International Conference on Interoperability for Enterprise Systems and Applications, 2016
  - SATTA: ACM SAC Conference Track on Software Architecture: Theory, Technology, and Applications at the ACM/SIGAPP Symposium On Applied Computing, 2016
  - AROSA: IEEE WETICE Conference Track on Adaptive and Reconfigurable Service-oriented and component-based Applications and Architectures, 2016
  - PESOS: International Workshop on Principles of Engineering Service-Oriented Systems at ACM/IEEE International Conference on Software Engineering (ICSE), 2016
  - WEA: ECSA International Workshop on Ecosystem Architectures, 2016
  - AFIN: International Conference on Advances in Future Internet, 2016
  - ADAPTIVE: International Conference on Adaptive and Self-Adaptive Systems and Applications, 2016
  - CAL: French Conference on Software Architecture, 2016
  - SBES: Brazilian Symposium on Software Engineering, 2016
  - SBCARS: Brazilian Symposium on Software Components, Architectures and Reuse, 2016
  - WDES: Workshop on Distributed Development of Software, Ecosystems and Systems-of-Systems, 2016
  - SESOS: ACM/IEEE ICSE International Workshop on Software Engineering for Systems-of-Systems, 2016
  - CSP: IEEE WETICE Conference Track on Collaborative Software Processes, 2016
  - COMPLEXIS: International Conference on Complexity, 2016

- Salah Sadou:
  - AROSA: IEEE WETICE Conference Track on Adaptive and Reconfigurable Service-oriented and Component-based Applications and Architectures, 2016
  - ICICS: International Conference on Information and Communication Systems, 2016
  - ASOCA: ICSOC Workshop on Adaptive Service-oriented and Cloud Applications, 2016
  - SESOS: ACM/IEEE ICSE International Workshop on Software Engineering for Systems-of-Systems, 2016
  - CIEL: French Conference on Software Engineering, 2016

7.5 Doctoral Examination Boards

- Flavio Oquendo:
  - Doctoral Examination Board (as Rapporteur) of Zakarea Al Shara (Migration des applications orientées objet vers celles à base de composants), LIRMM - Université de Montpellier, November 2016
- Salah Sadou:
  - Doctoral Examination Board (as Examiner) of Tarek Zernadji, Université de Biskra (Algeria), February 2016
  - Doctoral Examination Board (as Co-supervisor) of Mohamed Lamine Kerdoudi, Université de Biskra (Algeria), May 2016
  - HDR Examination Board (as Examiner) of Christelle Urtado, Université de Montpellier, Novembre 2016
- Isabelle Borne:
  - Doctoral Examination Board (President) of Francisco Javier Acosta Padilla (Self-adaptation for Internet of Things Applications), Université Rennes 1, 12 December 2016
  - Doctoral Examination Board (President) of David Fernando Méndez Acuña (Leveraging Software Product Lines Engineering in the Construction of Domain Specific Languages), Université Rennes 1, 12 December 2016
- Jérémy Buisson:
  - Doctoral Examination Board (as Examiner) of Seidali Rehab, Université de Constantine, 2015

7.6 Scientific Networks

- Flavio Oquendo: Co-chair of GT SdS (Systems-of-Systems)- GDR GPL

7.7 Industrial Collaborations

- Flavio Oquendo: Collaboration with MGDIS
- Flavio Oquendo: Collaboration with SEGULA
- Salah Sadou: Collaboration with SEGULA

7.8 Visiting Positions

7.9 Research and Doctoral Supervizing Awards (PEDR)

- Flavio Oquendo: PEDR (2016-2020)
- Salah Sadou: PEDR (2014-2018)

7.10 Laboratory Responsabilities

- Isabelle Borne: Responsible of the Site of Vannes for IRISA

7.11 Teaching Responsabilities

- Salah Sadou: Head of Computing Degree of ENSIBS School of Engineering (UBS, from September 2014)

7.12 Scientific Committees

- Flavio Oquendo:
  - Scientific Expert acting as reviewer and evaluator of R&D Projects for the European Commission (Horizon H2020) for:
    - Software-intensive Systems Engineering,
    - Systems-of-Systems, and
    - Cybersecurity & Trustworthy ICT.
  - Expert acting as evaluator of R&D Projects for the ANR (Agence Nationale de la Recherche) on Software Sciences and Technologies
- Salah Sadou:
  - Member of the Selection and Validation Committee (CSV) of Pôle Images et Réseaux
  - Member of Board DIS4 for ARED grants of PhD theses of Brittany Region

7.13 Academic Council (CAC)

- Salah Sadou: Member of the CAC (Commission recherche du conseil académique) of UBS

8 Bibliography

Major publications by the team in the current year

Books, Proceedings and Special Issues

[1] I. Bouassida Rodriguez, S. Kallel, F. Oquendo, *Special Issue on Adaptive and Reconfigurable Service-Oriented and Component-Based Applications*, *International Journal of Autonomous and Adaptive Communications Systems (IJAACS)*, *9*, 3/4, Inderscience, December 2016, `https://hal.archives-ouvertes.fr/hal-01441195`.

[2] F. Oquendo, M. Ali Babar, K. Drira, A. Legay, *Proceedings of the European Colloquium on Software-intensive Systems-of-Systems (ECSoS 2016)*, ACM DL, Copenhagen, Denmark, December 2016, `https://hal.archives-ouvertes.fr/hal-01445339`.

[3] F. Oquendo, P. Avgeriou, C. E. Cuesta, K. Drira, J. C. Maldonado, E. Y. Nakagawa, A. Zisman, *Proceedings of the 4th ACM/IEEE International Workshop on Software Engineering for Systems-of-Systems (SESoS 2016)*, Austin, Texas, United States, May 2016, https://hal.archives-ouvertes.fr/hal-01441206.

[4] F. Oquendo, K. Drira, A. Legay, T. Batista, *Proceedings of the 1st ACM SAC Conference Track on Software-intensive Systems-of-Systems (SiSoS 2017)*, ACM, Marrakesh, Morocco, April 2017, https://hal.archives-ouvertes.fr/hal-01445350.

[5] F. Oquendo, J. Leite, T. Batista, *Software Architecture in Action*, Springer, October 2016, https://hal.archives-ouvertes.fr/hal-01440285.

Doctoral Dissertations and "Habilitation" Theses

[6] E. R. De Sousa Cavalcante, *A Formally Founded Framework for Dynamic Software Architectures*, Theses, Université de Bretagne Sud / Université Bretagne Loire ; IRISA, June 2016, https://tel.archives-ouvertes.fr/tel-01426029.

[7] M. B. Gonçalves, *Supporting architectural design of acknowledged Software-intensive Systems-of-Systems*, Theses, Université de Bretagne-Sud / Université Bretagne Loire ; IRISA, December 2016, https://hal.archives-ouvertes.fr/tel-01441575.

[8] S. Martinez, *Plates-formes et mises à jour dynamiques configurables*, PhD Thesis, Télécom Bretagne / Université Bretagne Loire ; IRISA, March 2016, https://hal.archives-ouvertes.fr/hal-01444142.

Articles in Refereed Journals and Book chapters

[9] L. Bueno Ruas De Oliveira, E. Leroux, K. Romero Felizardo, F. Oquendo, E. Yumi Nakagawa, "ArchSORS: A Software Process for Designing Software Architectures of Service-Oriented Robotic Systems", *The Computer Journal (Oxford)*, Accepted in 2016 - To Appear in 2017, https://hal.archives-ouvertes.fr/hal-01442597.

[10] M. L. Kerdoudi, C. Tibermacine, S. Sadou, "Opening Web Applications for Third Party Development: a Service-Oriented Solution", *Service Oriented Computing and Applications*, February 2016, https://hal-lirmm.ccsd.cnrs.fr/lirmm-01276797.

[11] C. Tibermacine, S. Sadou, T. M. Ton That, C. Dony, "Software Architecture Constraint Reuse-by-Composition", *Future Generation Computer Systems*, February 2016, https://hal-lirmm.ccsd.cnrs.fr/lirmm-01276796.

[12] T. M. Ton That, S. Sadou, F. Oquendo, R. Fleurquin, "Preserving Architectural Decisions through Architectural Patterns", *Journal of Automated Software Engineering 23*, 3, September 2016, p. 427–467, https://hal.archives-ouvertes.fr/hal-01440366.

Publications in Conferences and Workshops

[13] A. Arnold, M. Baleani, A. Ferrari, M. Marazza, V. Senni, A. Legay, J. Quilbeuf, C. Etzien, "An Application of SMC to continuous validation of heterogeneous systems", *in: Simutools 2016 - Ninth EAI International Conference on Simulation Tools and Techniques*, Prague, Czech Republic, August 2016, https://hal.inria.fr/hal-01390487.

[14] E. Cavalcante, N. Cacho, F. Lopes, T. Batista, F. Oquendo, "Thinking Smart Cities as Systems-of-Systems: A Perspective Study", *in: Proceedings of the 2nd International Workshop on Smart Cities (SmartCities 2016) at ACM/IFIP/USENIX Middleware 2016*, ACM, p. 9:1–9:4, Trento, Italy, December 2016, `https://hal.archives-ouvertes.fr/hal-01441467`.

[15] E. Cavalcante, J. Quilbeuf, L.-M. Traonouez, F. Oquendo, T. Batista, A. Legay, "Statistical Model Checking of Dynamic Software Architectures", *in: ECSA 2016 - 10th European Conference on Software Architecture, Proceedings of the 2016 European Conference of Software Architecture*, Copenhague, Denmark, November 2016, `https://hal.inria.fr/hal-01390707`.

[16] T. P. Correia, M. Guessi, L. Bueno Ruas De Oliveira, E. Y. Nakagawa, "RARep: a Reference Architecture Repository", *in: 28th International Conference on Software Engineering and Knowledge Engineering (SEKE 2016)*, p. 363–368, Redwood, United States, July 2016, `https://hal.archives-ouvertes.fr/hal-01432702`.

[17] L. Garcés, F. Oquendo, E. Yumi Nakagawa, "A Quality Model for AAL Software Systems", *in: IEEE 29th International Symposium on Computer-Based Medical Systems (CBMS)*, p. 175–180, Belfast, Nothern Ireland, United Kingdom, June 2016, `https://hal.archives-ouvertes.fr/hal-01429106`.

[18] M. Guessi, F. Oquendo, E. Y. Nakagawa, "Checking the architectural feasibility of Systems-of-Systems using formal descriptions", *in: 11th System of Systems Engineering Conference (SoSE)*, p. 1–6, Kongsberg, Norway, June 2016, `https://hal.archives-ouvertes.fr/hal-01432700`.

[19] S. Mesli, A. Bignon, D. Kesraoui, A. Toguyeni, F. Oquendo, P. Berruet, "Vérification formelle de chaines de contrôle-commande d'éléments de conception standardisés", *in: Proceedings of the 11th International Conference on Modeling, Optimization & Simulation (MOSIM 2016)*, Montréal, Canada, August 2016, `https://hal.archives-ouvertes.fr/hal-01441589`.

[20] S. Mesli, D. Kesraoui, F. Oquendo, A. Bignon, A. Toguyéni, P. Berruet, "Formal Verification of Software-Intensive Systems Architectures Described with Piping and Instrumentation Diagrams ", *in: Proceedings of the 10th European Conference on Software Architecture (ECSA 2016)*, *LNCS*, 9839, Springer, p. 210–226, Copenhagen, Denmark, November 2016, `https://hal.archives-ouvertes.fr/hal-01440744`.

[21] S. Mesli, A. Toguyéni, A. Bignon, F. Oquendo, D. Kesraoui, P. Berruet, "Formal and Joint Verification of Control Programs and Supervision Interfaces for Socio-technical Systems Components", *in: Proceedings of the 13th IFAC Symposium on Analysis, Design, and Evaluation of Human-Machine Systems (HMS 2016)*, IFAC, p. 427–467, Kyoto, Japan, August 2016, `https://hal.archives-ouvertes.fr/hal-01441587`.

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

[23] F. Oquendo, J. Buisson, E. Leroux, G. Moguérou, J. Quilbeuf, "The SoS Architect Studio: Toolchain for the Formal Architecture Description and Analysis of Software-intensive Systems-of-Systems with SosADL", *in: Proceedings of the European Colloquium on Software-intensive Systems-of-Systems (ECSoS)*, Copenhagen, Denmark, November 2016, `https://hal.archives-ouvertes.fr/hal-01443130`.

[24] F. Oquendo, J. Leite, T. Batista, "Executing Software Architecture Descriptions with SysADL", *in: Proceedings of the 10th European Conference on Software Architecture (ECSA 2016)*, *LNCS*, 9839, Springer, p. 129–137, Copenhagen, Denmark, November 2016, `https://hal.archives-ouvertes.fr/hal-01440546`.

[25] F. Oquendo, J. Leite, T. Batista, "Specifying Architecture Behavior with SysADL", *in: 13th Working IEEE/IFIP Conference on Software Architecture (WICSA 2016)*, Venice, Italy, April 2016, `https://hal.archives-ouvertes.fr/hal-01433555`.

[26] F. Oquendo, "Case Study on Formally Describing the Architecture of a Software-intensive System-of-Systems with SosADL", *in: 15th IEEE International Conference on Systems, Man, and Cybernetics (SMC 2016)*, Budapest, Hungary, October 2016, `https://hal.archives-ouvertes.fr/hal-01433364`.

[27] F. Oquendo, "Formally Describing the Architectural Behavior of Software-Intensive Systems-of-Systems with SosADL", *in: Proceedings of the 21st IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 2016)*, IEEE, p. 13–22, Dubai, United Arab Emirates, November 2016, `https://hal.archives-ouvertes.fr/hal-01443173`.

[28] F. Oquendo, "Formally describing the software architecture of Systems-of-Systems with SosADL", *in: 11th IEEE System of Systems Engineering Conference (SoSE)*, Kongsberg, Norway, June 2016, `https://hal.archives-ouvertes.fr/hal-01433103`.

[29] F. Oquendo, "π-Calculus for SoS: A foundation for formally describing software-intensive Systems-of-Systems", *in: 11th IEEE System of Systems Engineering Conference (SoSE)*, IEEE, p. 1–6, Kongsberg, Norway, June 2016, `https://hal.archives-ouvertes.fr/hal-01433093`.

[30] F. Oquendo, "π-Calculus for SoS: A Novel π-Calculus for the Formal Modeling of Software-intensive Systems-of-Systems", *in: 38th International Conference on Communicating Process Architectures (CPA 2016)*, Copenhagen, Denmark, August 2016, `https://hal.archives-ouvertes.fr/hal-01433127`.

[31] F. Oquendo, "Software Architecture Challenges and Emerging Research in Software-Intensive Systems-of-Systems", *in: Proceedings of the 10th European Conference on Software Architecture (ECSA 2016)*, Springer (editor), *LNCS*, 9839, Springer, p. 3–21, Copenhagen, Denmark, November 2016, `https://hal.archives-ouvertes.fr/hal-01440536`.

[32] F. Petitdemange, I. Borne, J. Buisson, "Assisting the evolutionary development of SoS with reconfiguration patterns", *in: Sustainable Architecture: Global Collaboration, Requirements, Analysis (SAGRA)*, ACM, p. 9, Copenhagen, Denmark, November 2016, `https://hal.archives-ouvertes.fr/hal-01421487`.

[33] J. Quilbeuf, E. Cavalcante, L.-M. Traonouez, F. Oquendo, T. Batista, A. Legay, "A Logic for the Statistical Model Checking of Dynamic Software Architectures", *in: ISoLA*, *Leveraging Applications of Formal Methods, Verification and Validation: Foundational Techniques*, *9952*, Springer, p. 806 – 820, Corfou, Greece, October 2016, `https://hal.inria.fr/hal-01387429`.

[34] E. Silva, E. Cavalcante, T. Batista, F. Oquendo, "Bridging Missions and Architecture in Software-intensive Systems-of-Systems", *in: Proceedings of the 21st IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 2016)*, IEEE, p. 201–206, Dubai, United Arab Emirates, November 2016, `https://hal.archives-ouvertes.fr/hal-01441440`.

Miscellaneous

[35] I. Bouassida Rodriguez, S. Kallel, F. Oquendo, “Guest Editorial of the Special Issue on Adaptive and Reconfigurable Service-oriented and Component-based Applications and Architectures of the International Journal of Autonomous and Adaptive Communications Systems (Inderscience)”, December 2016, International Journal of Autonomous and Adaptive Communications Systems (IJAACS), Vol. 9, No. 3/4, `https://hal.archives-ouvertes.fr/hal-01114153`.