Research-Team ArchWare

Software Architecture

IRISA Site: Vannes

Activity Report

2015

1 Team

Head of the team

Flavio Oquendo, Full Professor, PES, 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
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, since December 2015

PhD students

Delphine Beaulaton, MESR grant, since October 2015
Youcef Bouziane, Cotutelle grant, since September 2014
Everton Cavalcante, CsF-CNPq grant, since October 2013
Imane Cherfa, Cotutelle grant, since September 2014
Marcelo Gonçalves, CsF-CNPq grant, since October 2013
Valdemar Graciano Neto, CsF-CNPq grant, since November 2015
Milena Guessi, FAPESP grant, since September 2014
Davy Hélard, CIFRE grant with MGDIS, defended his thesis on December 2015
Soraya Mesli-Kesraoui, CIFRE grant with SEGULA, since November 2013
Lucas Oliveira, CsF-CNPq grant, defended his thesis on June 2015
Franck Petitdemange, MESR grant, since September 2014

Lina Rodriguez, FAPESP grant, since December 2015
Abderrhamane Seriai, ARED grant w/U. Montreal, defended his thesis on January 2015
Sébastien Martinez, ARED grant with Télécom Bretagne, since October 2012 (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 deriving/generating abstract/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 π-Calculus for Systems-of-Systems

Keywords: π-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 continued the definition of a type system. We have followed the common approach, organizing the type system based on the structure of the language, as defined by the underlying meta-model.

The work is largely in progress, under development using the Coq proof assistant. Among the issues, evidences must be provided that the (non-formal, Java-based) implementation conforms to the formal type system. In order to address this issue, we investigated an approach deriving from proof-carrying code. We plan to design the compiler such that it can generate a proof of well-typedness in addition to type information.

3.4 Intermediate Model for SoS Analysis on SosADL

Keywords: Formal Analysis, Transition Systems, Software Architecture, Systems-of-Systems (SoS).

Participants: Elena Leroux, Gersan Moguérou, Flavio Oquendo.

To deal with the complexity of SoSs, it is necessary to design its architecture while allowing (1) to abstract away the details of the constituent systems of the SoS; (2) to represent the SoS as sets of constituent systems connected through mediators enforcing their interactions (a) among the constituent systems, and (b) between the constituent systems and the environment; and (3) to guide the SoS design and evolution. A key issue is to enable the analysis of such complex architectures regarding specified properties. To make possible SoS architecture analysis, we propose to represent the behavioral semantics of each SoS constituent system as a formal model, called ioSTS (input-output Symbolic Transitions System). ioSTS is a variant of a Labeled Transitions Systems widely used by different techniques such as model checking, conformance testing, simulation etc. Thereby, this translation enables the use of existing tools such as Uppaal and STG to perform different analyses.

We have defined the ioSTS representation of SosADL as the denotational semantics of SosADL focusing on the behavioral specification of constituents and their composition in coalitions. The non-behavioral features of SosADL are factorized between the two formalisms.

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.

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.

The realization of SoS architecture descriptions faces particular challenges related to dynamic evolution, interface mismatches, and quality characteristics at the SoS abstraction level. In particular, Architecture Description Languages (ADLs) have become well-accepted approaches for systematically representing and analyzing software architectures. One of the most emergent ADLs for describing SoS is SosADL, a language that supports abstract descriptions of constituents, mediators, and their architectural configuration (i.e., the arrangement of constituents and mediators that form an SoS) amongst other desired characteristics of ADLs for SoS.

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). In particular, this approach formally

defines the constraints that govern the organization of constituents and mediators at runtime, such as constraints preventing the formation of coalitions with isolated constituent systems. The formalization of architectural configurations in terms of a CSP enables is interesting for taking advantage of existing tools for automatically finding concrete configurations that meet their abstract specifications.

We formalized this approach using Alloy, a formal language for expressing complex structural constraints and behaviors. We developed an Alloy formal metamodel for SosADL that captures its main architectural elements, e.g., constituent systems, mediators, and coalitions, as well as dependencies among them. Thus, our approach can take advantage of the Alloy Analyzer, which supports the generation of instances of model invariants, the simulation of operations defined as part of a model, as well as the confirmation of user-specified properties of an SoS model. As a result, the execution of an Alloy model in this tool will attempt to find a solution for the CSP, which in our approach corresponds to a concrete configuration of an SoS that satisfies its given abstract description, based on available concrete constituent systems.

3.6 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.

3.7 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.

3.8 Architectural Design of Service-oriented Robotic Systems and Systems-of-Systems

Keywords: Service Oriented Architecture, 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.

3.9 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.

3.10 Supporting Dynamic Reconfiguration of Software-intensive Systems with Coqcots & Pycots

Keywords: Dynamic Reconfiguration, Component Based Systems, Software Architecture.

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

Dynamic reconfiguration of component-based software systems is well-established. Based on quiescence, the mainstream approach consists in suspending and resuming selected components. In practice this approach requires suspending the components that depends on suspended components, so the suspension propagates through the whole architecture. The most common alternative approach, such as OSGi, consists in considering all the dependencies as optional. This approach is impractical too because component implementations are expected check for the dependencies and must provide some behavior in case the dependencies are not satisfied.

To solve this issue, we propose to use Dynamic Software Updating in order to change the

implementation and type of components at runtime. That way, the behavior of the components can be changed temporarily to accommodate missing dependencies during the time of reconfiguration.

The semantics of the reconfiguration actions has been mechanized using the Coq proof assistant. This work turns the Coq language into a reconfiguration language. On the one side, the proof assistant allows to formally verify the reconfiguration. For instance proof obligations are issued for the preconditions of each reconfiguration action. On the other side, the extraction mechanism of Coq is used to generate the executable reconfiguration script.

3.11 Categorization of Dynamic Software Updating Mechanisms

Keywords: Dynamic Software Updating (DSU).

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

While traditionally Dynamic Software Updating (DSU) comes as a platform the application is adapted to at design time in order to enable updates, such assumption cannot be made anymore in the context of complex software systems, composed of several components designed independently one of the others. It is therefore crucial to better understand the basic mechanisms and their interactions.

We have proposed a framework for the study of DSU platform. Given this framework, we performed a statistical study of the literature platforms and mechanisms involving mining and clustering methods. As a result, we identified similarities between platforms as well as existing combinations of mechanisms. This works helped us in characterizing the interactions and dependencies between the mechanisms at the core of each platform.

In relation to this work based on the literature, we have also proposed a formal operational semantics for updates. Our approach consists in defining a general formal language, supporting primitive mechanisms for the modification of an executing program. The language supports mainly the reception of an update, its scheduling, the change of the execution flow, and the change of values. We have successfully used this language to model several DSU platforms from the literature.

3.12 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.

3.13 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.

Programmers and designers traditionally focus on the performance and functional correctness of the software that they are producing. Performance and correctness are not sufficient in an industrial context, where the software should also foster security. Security has received a growing attention over the last decade and is now a major concern, but is still not systematically taken into account in the design of software-intensive systems.

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.

3.14 Development of an Architectural Framework for Business Intelligence

Keywords: Service-oriented architecture, Business intelligence, Budget planning.

Participants: Davy Hélard, Jean-Philippe Gouigoux (MGDIS), Flavio Oquendo.

In the scope of Business Intelligence, planning aims to support multiple actors in their process of converging different views and problematics from different domains to get a shared business planning model. It is in particular the case of business planning in local government.

A major difficulty in business planning is that each actor states her/his views and problematics with a different time scale. Integrating them into a unique model that represents a

common state of reality becomes very costly and awkward to manage when basing the construction of these models on discrete modeling techniques used by current tools of business planning. This research proposes a novel solution, beyond the state-of-the-art, for addressing these issues: it conceives a novel metamodel based on a continuous time calculus. Through the developed approach, it allows multiple actors to integrate the different business logics of their planning domain in a shared model as well as to observe it from different time scales. The advantages of our solution based on continuous time against solutions based on discrete time are presented through a case study.

The conceived metamodel was implemented within a real industrial set in MGDIS (a company specialized in business planning for local governments) following an innovative service oriented architecture: this architecture segregates the modeling from the evaluation to allow the parallelization of model evaluation for big volumes of data. Besides case studies, it was validated by MGDIS experts on business planning against real requirements.

The consistency is enforced by a novel service-oriented architecture that segregates the model and the different evaluations, supporting concurrent executions from different viewpoints.

3.15 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.

3.16 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.

The concept of SoS is widespread. However, the volatility of systems compared to components, calls into question the stability of the SoS architecture. Indeed, the latter must perpetually adapt to changes in the environment of the SoS. Moreover, an SoS exists to fulfill a specified mission. And the specification of the latter is relatively stable compared to the architecture that implements the SoS. Based on these assumptions, we propose a Mission Description Language (MDL) to define SoS missions. The definition of the mission of an SoS has a twofold objective: (i) to provide SoS engineers with a stable documentation of the SoS in order to manage its evolutions and (ii) to give the ability to automatically generate the SoS architecture from the mission definition. This MDL was defined through a litterature study where we collected concepts covering the notion of mission in different application fields. The meta-model of the defined MDL represents the synthesis of these concepts.

3.17 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.

One of the SoS properties is the dynamic adaptation to changes in their environment. This implies the need to frequently, on-the-fly, rearchitect the SoS. This reachitecture is driven by the defined mission.

Thus, we propose an approach of architecting SoSs based on the definition of the mission. The definition of the mission is based on the principle of situation/reaction with an implied architecture. We propose a definition model where the dynamic aspect of the SoS environment is implicit. The situations are defined by an expert system and reactions by orchestrations.

4 Software

4.1 SoStudio: The SoS Architect Studio for SosADL

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

SoStudio 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: SoStudio Support for Verification

Participants: Elena Leroux, Gersan Moguérou.

SosADL2ioSTS is a part of the SoStudio 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 implementation has started, but needs to be integrated with the type checker, under development.

4.3 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.

4.4 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.

4.5 PiADL2Go: Supporting Dynamic Software Architectures from π-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 - 2015
- 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 mission-based dynamic software-intensive systems-of-systems (PhDs in co-tutelle)
- Salah Sadou:
  - University of Montréal (Houari Sahraoui): Restructuring object-oriented systems into component-based systems (PhD in co-tutelle)
  - University of Science and Technology of Houari Boumedienne, Alger, Algeria (Mohamed Ahmed Nacer): Product line architecture for systems-of-systems (PhD in co-tutelle)
  - 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)

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)

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 Advanced Architectures for the Future Generation of Software-intensive Systems of the Elsevier Journal on Future Generation Computer 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)

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), Florence, Italy, 2015 (Program Chair)
  - CBSOFT Workshop on Distributed Development of Software, Ecosystems and Systems-of-Systems (WDES), Belo Horizonte, Brazil, 2015
- Salah Sadou:
  - Advanced in Software Engineering Track of the ACS/IEEE International Conference on Computer Systems and Applications (AICCSA), Marrakech, Maroc, 2015 (Program Chair)

  - French Conference on Software Engineering (CIEL), Bordeaux, France, 2015 (Program Chair)

7.4 Program Committees

- Isabelle Borne:
  - SESOS: ACM/IEEE ICSE International Workshop on Software Engineering for Systems-of-Systems, 2015
  - AROSA: IEEE WETICE Conference Track on Adaptive and Reconfigurable Service-oriented and Component-based Applications and Architectures, 2015
- Jérémy Buisson:
  - ICCS: International Conference on Computational Science, 2015
  - C&ESAR: Computer & Electronics Security Applications Rendez-vous, 2015
- Régis Fleurquin
  - Advanced in Software Engineering Track of ACS/IEEE AICCSA, 2015
- Flavio Oquendo:
  - WICSA: IEEE/IFIP Working International Conference on Software Architecture, 2015
  - ECSA: European Conference on Software Architecture, 2015
  - WCCS: World Conference on Complex Systems, 2015
  - ICSEA: International Conference on Software Engineering Advances, 2015
  - I-ESA: International Conference on Interoperability for Enterprise Systems and Applications, 2015
  - SERA: IEEE/ACIS International Conference on Software Engineering Research, Management and Applications, 2015
  - SATTA: ACM SAC Conference Track on Software Architecture: Theory, Technology, and Applications at the ACM/SIGAPP Symposium On Applied Computing, 2015
  - AROSA: IEEE WETICE Conference Track on Adaptive and Reconfigurable Service-oriented and component-based Applications and Architectures, 2015
  - PESOS: International Workshop on Principles of Engineering Service-Oriented Systems at ACM/IEEE International Conference on Software Engineering (ICSE), 2015
  - IWSECO-WEA: Joint International Workshop on Software Ecosystems & International Workshop on Ecosystem Architectures, 2015
  - COBRA: International Workshop on Exploring Component-based Techniques for Constructing Reference Architectures, 2015

  - AFIN: International Conference on Advances in Future Internet, 2015
  - ICSOFT-PT: International Conference on Software Paradigm Trends, 2015
  - ADAPTIVE: International Conference on Adaptive and Self-Adaptive Systems and Applications, 2015
  - CAL: Conférence Francophone sur les Architectures Logicielles, 2015
  - SBES: Brazilian Symposium on Software Engineering, 2015
  - SBCARS: Brazilian Symposium on Software Components, Architectures and Reuse, 2015
  - WDES: Workshop on Distributed Development of Software, Ecosystems and Systems-of-Systems, 2015
  - SESOS: ACM/IEEE ICSE International Workshop on Software Engineering for Systems-of-Systems, 2015

- Salah Sadou:

  - AROSA: IEEE WETICE Conference Track on Adaptive and Reconfigurable Service-oriented and Component-based Applications and Architectures, 2015
  - SERA: IEEE/ACIS International Conference on Software Engineering Research, Management and Applications, 2015 SESOS: ACM/IEEE ICSE International Workshop on Software Engineering for Systems-of-Systems, 2015
  - QUORS: IEEE International Workshop on Quality Oriented Reuse of Software, 2015
  - CAL: Conférence Francophone sur les Architectures Logicielles, 2015
  - MCETECH: International Conference on E-Technologies, 2015

7.5 Doctoral Examination Boards

- Flavio Oquendo:

  - Doctoral Examination Board (as Rapporteur) of Cédric Eichler (Modélisation de systèmes dynamiques autonomes : graphe, réécriture et grammaire), LAAS - Université de Toulouse-Paul Sabatier, June 2015
  - Doctoral Examination Board (as President) of Anas Shatnawi (Supporting Reuse by Reverse Engineering Software Architectures and Components from Object-Oriented Product Variants and APIs), LIRMM - Université de Montpellier, June 2015

- Salah Sadou:

  - Doctoral Examination Board (as Examiner) of Abderrahman Mokni, LIRMM - Université de Montpellier, October 2015
  - Doctoral Examination Board (as Examiner) of Okba Tibermacine, Université de Biskra (Algérie), June 2015

- Isabelle Borne:
  - Doctoral Examination Board (as President) of Abderrahmane Seriai (Rendre réutilisable des composants extraits d'une application orientée objets), IRISA - Université de Bretagne-Sud, January 2015
  - Doctoral Examination Board (as President) of Paola Vallejo (Réutilisation de composants logiciels pour l'outillage de DSML dans le contexte des MPSoC), Lab-STICC - Université de Bretagne Occidentale, December 2015
- Jérémy Buisson:
  - Doctoral Examination Board (as Examiner) of Seidali Rehab, Université de Constantine, 2015

7.6 Scientific Networks

- Isabelle Borne: Co-chair of Action IDM (Model-Driven Engineering) - GDR GPL & ASR
- 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 Research Excellence Awards (PES)

- Flavio Oquendo: PES A (2011-2015)
- Salah Sadou: PEDR (2014-2018)

7.9 Laboratory Responsabilities

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

7.10 Teaching Responsabilities

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

7.11 Scientific Committees

- Salah Sadou: Member of the Selection and Validation Committee (CSV) of Pôle Images et Réseaux
- Salah Sadou: Member of Board DIS4 for ARED grants of PhD theses of Brittany Region

7.12 Expertises

- Flavio Oquendo: 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.

7.13 Academic Council (CA), National Council of Universities (CNU)

- Isabelle Borne: Member of CNU (Conseil national des universités), Section 27
- 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, Special Issues and Monographs

[1] P. Avgeriou, C. E. Cuesta, K. Drira, E. Y. Nakagawa, J. C. Maldonado, F. Oquendo, A. Zisman, *Proceedings of the 3rd ACM/IEEE International Workshop on Software Engineering for Systems-of-Systems (SESoS 2015)*, ACM, Florence, Italy, May 2015, `https://hal.archives-ouvertes.fr/hal-01441064`.

[2] K. Drira, F. Oquendo, *Special Issue on Advanced Architectures for the Future Generation of Software-Intensive Systems*, *International Journal on Future Generation Computer Systems (FGCS)*, *47*, June 2015, `https://hal.archives-ouvertes.fr/hal-01440508`.

Doctoral Dissertations and "Habilitation" Theses

[3] L. Bueno Ruas De Oliveira, *Architectural design of service-oriented robotic systems*, Theses, Université de Bretagne Sud / Université Européenne de Bretagne ; IRISA, June 2015, `https://tel.archives-ouvertes.fr/tel-01299803`.

[4] D. H. Helard, *A metamodel calculation continuous time for aid decision systems applied to financial planning*, Theses, Université de Bretagne Sud / Université Européenne de Bretagne ; IRISA, December 2015, `https://tel.archives-ouvertes.fr/tel-01316887`.

[5] A. Seriai, *Rendre réutilisables des composants extraits d'une application orientée-objet*, PhD Thesis, Université de Bretagne Sud / Université Européenne de Bretagne ; IRISA, January 2015, `https://hal.archives-ouvertes.fr/hal-01444169`.

Articles in Refereed Journals and Book Chapters

[6] L. Bueno Ruas De Oliveira, F. Amaral, D. Martins, F. Oquendo, E. Y. Nakagawa, "RoboSeT: A Tool to Support Cataloging and Discovery of Services for Service-Oriented Robotic Systems", *Communications in Computer and Information Science*, December 2015, p. 114–132, https://hal.archives-ouvertes.fr/hal-01441259.

[7] J. Buisson, F. Dagnat, E. Leroux, S. Martinez, "Safe reconfiguration of Coqcots and Pycots components", *Journal of Systems and Software*, 2015, https://hal.archives-ouvertes.fr/hal-01235602.

[8] A. Legay, J. Quilbeuf, F. Oquendo, "Verifying Systems-of-Systems with Statistical Model Checking", *ERCIM News*, 103, 2015, https://hal.inria.fr/hal-01242652.

[9] F. Oquendo, A. Legay, K. Drira, "GT SoS: Research Network on Trustworthy Software-intensive Systems-of-Systems", *ERCIM News*, 102, 2015, https://hal.inria.fr/hal-01242651.

[10] F. Oquendo, A. Legay, "Formal Architecture Description of Trustworthy Systems-of-Systems with SosADL", *ERCIM News*, 102, 2015, https://hal.inria.fr/hal-01242649.

[11] T. M. Ton That, S. Sadou, F. Oquendo, I. Borne, "Preserving architectural pattern composition information through explicit merging operators", *Future Generation Computer Systems 47*, June 2015, p. 97–112, https://hal.archives-ouvertes.fr/hal-01440346.

Publications in Conferences and Workshops

[12] G. Abdalla, C. D. N. Damasceno, M. Guessi, F. Oquendo, E. Y. Nakagawa, "A Systematic Literature Review on Knowledge Representation Approaches for Systems-of-Systems", *in : 9th Brazilian Symposium on Components, Architectures and Reuse Software (SBCARS 2015)*, p. 70–79, Belo Horizonte, Brazil, September 2015, https://hal.archives-ouvertes.fr/hal-01432705.

[13] E. Cavalcante, T. Batista, F. Oquendo, "Supporting Dynamic Software Architectures: From Architectural Description to Implementation", *in : Proceedings of the 12th Working IEEE/IFIP Conference on Software Architecture (WICSA 2015)*, IEEE, p. 31–40, Montreal, Canada, May 2015, https://hal.archives-ouvertes.fr/hal-01441459.

[14] B. Costa, P. F. Pires, F. C. Delicato, F. Oquendo, "Towards a View-Based Process for Designing and Documenting RESTful Service Architectures", *in : Proceedings of the 2015 European Conference on Software Architecture Workshops*, 50, Dubrovnik/Cavtat, Croatia, September 2015, https://hal.archives-ouvertes.fr/hal-01440890.

[15] M. Gonçalves, F. Oquendo, E. Y. Nakagawa, "A meta-process to construct software architectures for system-of-systems", *in : Proceedings of the 30th Annual ACM Symposium on Applied Computing (SAC 2015)*, ACM, Salamanca, Spain, April 2015, https://hal.archives-ouvertes.fr/hal-01441097.

[16] M. Gonçalves, F. Oquendo, E. Yumi Nakagawa, "A Meta-Process to Construct SoS Software Architectures", *in : 30th ACM Symposium on Applied Computing (ACM/SAC'2015)*, p. 1–6, Salamanca, Spain, April 2015, https://hal.archives-ouvertes.fr/hal-01113366.

[17] M. Guessi, L. Bueno Ruas De Oliveira, L. Garcés, F. Oquendo, "Towards a Formal Description of Reference Architectures for Embedded Systems", *in : 1st International Workshop on Exploring Component-based Techniques for Constructing Reference Architectures (CobRA 2015)*, p. 17–20, Montreal, Canada, May 2015, https://hal.archives-ouvertes.fr/hal-01432704.

[18] M. Guessi, E. Cavalcante, L. Bueno Ruas De Oliveira, "Characterizing Architecture Description Languages for Software-Intensive Systems-of-Systems", *in : 2015 IEEE/ACM 3rd International Workshop on Software Engineering for SystemsofSystems (SESoS)*, p. 12–18, Florence, Italy, May 2015, https://hal.archives-ouvertes.fr/hal-01432703.

[19] M. Guessi, D. A. Moreira, G. Abdalla, F. Oquendo, E. Yumi Nakagawa, "OntolAD: a Formal Ontology for Architectural Descriptions", *in : 30th ACM Symposium on Applied Computing (ACM/SAC'2015)*, p. 1–8, Salamanca, Spain, April 2015, https://hal.archives-ouvertes.fr/hal-01113369.

[20] M. Guessi, V. Neto, T. Bianchi, K. Romero Felizardo, F. Oquendo, E. Yumi Nakagawa, "A Systematic Literature Review on the Description of Software Architectures for Systems of Systems", *in : 30th ACM Symposium on Applied Computing (ACM/SAC'2015)*, p. 1–8, Salamanca, Spain, April 2015, https://hal.archives-ouvertes.fr/hal-01113368.

[21] F. Petitdemange, I. Borne, J. Buisson, "Approach Based Patterns for System-of-Systems Reconfiguration", *in : International Workshop on Software Engineering for Systems-of-Systems*, Florence, Italy, May 2015, https://hal.archives-ouvertes.fr/hal-01142539.

[22] F. Petitdemange, J. Buisson, I. Borne, "Une Approche Orientée Pattern pour la Reconfiguration de Système de Systèmes", *in : CIEL 2015*, Bordeaux, France, June 2015, https://hal.archives-ouvertes.fr/hal-01421540.

[23] J. Portocarrero, F. C. Delicato, P. F. Pires, E. Nakagawa, F. Oquendo, "Self-Adaptive Middleware for Wireless Sensor Networks: A Reference Architecture", *in : Proceedings of the 2015 European Conference on Software Architecture Workshops*, 12, ACM, Dubrovnik/Cavtat, Croatia, September 2015, https://hal.archives-ouvertes.fr/hal-01440881.

[24] E. Silva, T. Batista, F. Oquendo, "A mission-oriented approach for designing system-of-systems", *in : Proceedings of the 10th IEEE System-of-Systems Engineering Conference (SoSE 2015)*, IEEE, IEEE, p. 346–351, San Antonio, Texas, United States, May 2015, https://hal.archives-ouvertes.fr/hal-01441152.

Miscellaneous

[25] K. Drira, F. Oquendo, "Guest Editorial of the Special Issue on Advanced Architectures for the Future Generation of Software-Intensive Systems", June 2015, International Journal on Future Generation Computer Systems (FGCS), Vol. 47, June 2015, https://hal.archives-ouvertes.fr/hal-01114152.

[26] E. Y. Nakagawa, F. Oquendo, P. Avgeriou, C. E. Cuesta, K. Drira, J. C. Maldonado, A. Zisman, "Foreword: Towards Reference Architectures for Systems-of-Systems", May 2015, Proceedings of the 3rd ACM/IEEE International Workshop on Software Engineering for Systems-of-Systems at ICSE 2015, Florence, Italy, https://hal.archives-ouvertes.fr/hal-01441043.