Research-Team ArchWare

Software Architecture

IRISA Site: Vannes

Activity Report

2013

1 Team

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

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

Université de Bretagne-Sud
Isabelle Borne, Full Professor
Jérémy Buisson, Assistant Professor
Régis Fleurquin, Associate Professor, HDR
Elena Leroux, Assistant Professor
Salah Sadou, Associate Professor, HDR
Gersan Moguérou, Research Engineer (part time: 80%)

Visitors
Thais Batista, Associate Professor, UFRN
Jair Leite, Associate Professor, UFRN
Elisa Nakagawa, Associate Professor, USP

PhD students
Everton Cavalcante, CsF-CNPq grant, since October 2013
Tahar Gherbi, Assistant, defence planned on September 2014
Marcelo Gonçalves, CsF-CNPq grant, since October 2013
Salma Hamza, ARED grant, defence planned on September 2014
Davy Hélard, CIFRE grant with MGDIS, since September 2012
Soraya Mesli, CIFRE grant with SEGULA, since November 2013
Lucas Oliveira, CsF-CNPq grant, since October 2012
Abderrhamane Seriai, ARED grant with University of Montreal, defence planned on September 2014
Minh Tu Ton That, MESR grant, defence planned on September 2014
Qin Xiong, ARED grant, since December 2010

Master students
Soraya Mesli, MRI/UBS, February-June 2013

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).

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. This composition forms a larger system that performs a mission not performable by one of the constituent systems alone, i.e. it creates an "emergent behaviour". But, if the architect of an SoS has authority on how the different systems work together, it has no direct control over each of the involved systems in the SoS. Component systems fulfil valid purposes in their own right, and continue to operate to fulfil those purposes if disassembled from the encompassing SoS. They are managed, in part, for their own purposes rather than the purposes of the whole. Moreover, a SoS is itself a system and possibly a constituent system of a much more complex SoS in its own right.

Indeed, since the dawn of computing, the complexity of software and the criticality 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 being part of networked systems in the present, to becoming systems of systems in the coming future .

In fact, nowadays almost all aspects of our lives and livelihoods depend on some sort of software-intensive system, especially when these systems are critical. This is the case of applications found in different areas as diverse as e.g. aerospace, aeronautics, automotive, energy, healthcare, manufacturing, transportation, civil infrastructures, entertainment, and consumer appliances; and applications that addresses societal needs as e.g. in environmental monitoring, distributed energy grids, emergency coordination, global traffic control, and smart cities. Therefore, the endeavour of constructing critical systems evolved from engineering complicated systems in the last century, to architecting critical SoS in this century. De facto, critical SoS became among the most complex of man-made creations and its very nature has intrinsic properties that are hard to address.

Moreover the upcoming generation of critical SoS 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. 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 fulfil its own mission) without violating the independence of its constituent systems that are autonomous, acting to fulfil 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 fulfil 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 behaviours:* from the collaboration of the participating systems may emerge new behaviours. Furthermore, these behaviours may be ephemeral because the systems composing the SoS evolve independently, which may impact the availability of these behaviours. Hence, the challenge is to architect and construct a SoS in a way that emergent behaviours and their subsequent evolution can be discovered and controlled. In the case of an open-world environment, one can add the following characteristics and challenges:

6. *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 summarised 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 fulfil its own mission.

7. *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.

8. *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.

In essence, the long-term research challenge raised by SoSs calls for a paradigm shift in Software Architecture: SoSs call for a novel paradigm, complementing the traditional use of architectures at design-time traditionally applied to closed-world systems by novel trustful approaches blurring the boundary between design-time and run-time supporting continuous correctness of open-world systems.

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 fulfil central purposes;

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

Regarding the state-of-the-art, 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 SoS, i.e. collaborative and virtual SoS.

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. suitable architectural abstractions for formulating the architecture and re-architecture of SoS;

2. an appropriate formalism and its underlying computational model to rigorously specify the architecture and re-architecture of SoS;

3. the supporting mechanisms to construct and manage SoSs driven by architecture descriptions, while resiliently enforcing their correctness, effectiveness, and efficiency;

4. the supporting mechanisms to evolve SoSs driven by architecture descriptions, while resiliently enforcing their correctness, effectiveness, and efficiency throughout the evolution.

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, transformation, 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 models-at-runtime in the service-oriented computing style;

4. to architecturally and technologically ground the dynamic evolution of SoS architectures around the concept of "anytime architecture".

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 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 active. 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 active architectural models at run-time.

- Analysis of SoS architectures: SoS architecture descriptions, by their formal nature, will support automated analysis with a view in particular to evaluating and predicting non-functional qualities of the modelled SoS. In our approach, we will make a significant progress beyond the state-of-the-art of SoS analysis by developing techniques and tools for the architecture-centric model-based analysis of SoS. It will support verification of different sorts of properties and interleaving of these properties, including structural (e.g. connectivity, topology), behavioural (e.g. safety, liveness, fairness), and quality properties (e.g. agility, performance). We will develop 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. Automated verification will include infinite-state model checking and abstract interpretation techniques for the critical parts of SoS, and testing for non-critical parts to ensure the correct functioning of SoS.

- Architecture-driven construction of SoSs: SoS architecture will drive the initial construction and subsequent evolutions of a SoS. Initial construction will be developed through refinement, where abstract SoS architectures are refined to "meet in the middle" with concrete selection and integration of discovered systems, consequently defining concrete SoS architectures. In our approach, significant progress beyond the state-of-the-art on SoS construction will be achieved by conceiving abstractions and mechanisms for expressing architecture transformations, where the application of these transformations will support refinement from abstract to concrete SoS architectures, taking into account discovered SoSs. Each transformation can be seen as an architecture rewrite in a formal rewriting system. These transformations are enforced to be refinements if preconditions are satisfied and proof obligations discarded. Concrete architectures are deployed to directly implement running SoSs in terms of middleware-based glue code.

- Evolution of SoS architectures and underlying SoSs: an evolving system must be continuously aware of its own behaviour and surrounding environment to execute efficiently and to react to new situations. In the ArchWare Research Project, we will develop SoS-specific concepts and mechanisms of software instrumentation, based on probes and gauges, to implement and support such a continuous feedback. We will utilise a set of probes to observe and quantify significant events as defined by SoS requirements. The probes will have the ability to be placed into running SoSs. Each SoS will use the input from the probes according to the interpretation of SoS-specific gauges, together with the models of the mission and the architecture to decide when, how, and where evolution is appropriate. For supporting feedback, due to the specific nature of a SoS, monitoring will be required at two distinct levels: the architecture of the SoS and its assigned mission. On the one hand, monitoring the architecture of the SoS implies maintaining a suitable model of all dependencies among the systems involved in the SoS. Those dependencies being of various types, we will formally describe them in such a way that any change in the architecture will be automatically detected. On the other hand, monitoring the mission implies collecting data from the systems comprising the SoS, indicating that the mission is in progress, or detecting any event that must be addressed to decide if one has to re-architect the SoS and/or change the mission itself. Due to the nature of the SoS, a challenge is that the mission monitoring system must be itself evolvable as systems involved in the SoS may evolve over time.

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

3 New Results

3.1 The SoSmart SoS Architecture Framework

Keywords: Architecture Description Language, Systems-of-Systems, Software Architecture.

Participants: Flavio Oquendo, Isabelle Borne, Jérémy Buisson, Régis Fleurquin, Elena Leroux, Salah Sadou, Gersan Moguérou.

The meta-model underpinning the SoSmart language has been defined. Its main concepts are: coalition, system and mediator. The architecture of an SoS is an evolving coalition of systems, bound by mediators.

Compared to previous architecture description languages (ADL), the SoSmart makes a clear distinction between:

- The coalition is the dynamic composition of the SoS constituents (systems and mediators). It is declared intentionally as a set of constraints on the number of constituents and on the binding between these constituents. It is concretized at runtime depending on the discovered systems. It evolves during operation.
- The systems model and reify the constituents taken from the environment. Depending on the category of SoS being considered, systems may be pre-existent or developed

specifically for the SoS. The systems are dynamically discovered by the SoS.

- The mediators provide some logic to adapt and translate the communication protocoles between systems that have been independently developed. Mediators are dynamically instantiated by the coalition, as needed.

As a result of this distinction, SoSmart relies on three different fragments of the process algebra for each of these elements.

The SoSmart language is under development, relying on this meta-model.

3.2 Architecture-based Conformance Testing

Keywords: Software Architecture, Architecture Analysis, Conformance Testing.

Participants: Elena Leroux, Qin Xiong, Flavio Oquendo.

Software architecture plays a central role in the development of software systems. It provides a high-level description for large-size, complex, software-intensive systems using suitable abstractions of the system's components and their interactions. In our work, the software architecture is described using a formal Architecture Description Language (ADL) designed in the ArchWare European Project, Pi-ADL-C&C. One of the purposes of this ADL is to allow formal validation of an implemented system with respect to its architectural model. In this work, we propose a conformance testing approach for validating a software system with respect to its architecture. The architectural abstract test cases are derived from an Input-Output Symbolic Transition System (IOSTS) representing the architecture structure and behaviors, which are then translated into concrete test cases to be executed on the system under test. In aim of this work was to examine the applicability of our architecture-based conformance testing approach on a simple exemple of the coffee machine. The idea is to extend this approach to software-intensive systems-of-systems, and to use the results, such as use of IOSTS model to represent a software architecture, for the formal architectural analysis of SoS.

3.3 Using Object-Oriented metrics on Component-Based Applications

Keywords: Software Architecture, Component Properties, OO Metrics, OSGI Framework, Software Quality.

Participants: Salma Hamza, Salah Sadou, Régis Fleurquin.

Component Based Software Development (CBSD) is an important paradigm and architectural style. The expected benefits are better reliability, reduced development costs and shorter life cycles. But to ensure that a component-based development meets these characteristics, applications and their components should respect some "good" properties. Many quality models have been proposed to help in defining the quality of component-based applications. The proposed models follow the old ISO 9126 quality model framework with few corrections and adjustments made to suit the component-based software domain aims. Thus, these models conform to a hierarchical model which formalizes the quality of a component in terms of its characteristics and sub-characteristics. Further, sub-characteristics are divided into attributes

that can be checked or measured. In a quality model, measures concern either internal aspects (source codes) or external aspects (application in use).

The internal metrics are, in particular, a good means for component developers and architects to predict the quality characteristics of their produced artefacts during coding and/or design stages. In the literature, there is some consensus that component-based metrics require a different approach than object-oriented metrics. Therefore, several internal metrics have been proposed specifically for the component paradigm. But: i) there is no consensus yet on many of the concepts and elements that are measured by these metrics, ii) inconsistencies in component definitions can frequently be found in many studies, iii) they lack a statistically significant experimental validation and a set of experimental data to provide better insight into their use. Last, but not the least, these metrics are not supported by any code analysis tools yet. Thus, developers and architects do not (and cannot) use these metrics. Without such internal metrics, an accurate planning and quality checking of the development process are more difficult to achieve.

Most of component models are build upon the object-oriented paradigm (such as EJB, OSGi, CCM, FRACTAL, etc.). Thus, to solve the problem above we decided to examine if some existing internal object-oriented metrics can help developers and architects to measure some of the essential structural properties that affect the quality of components and component-based applications. Indeed, these metrics have the advantage of being well-defined, subject of numerous empirical validation and so well-understood, and especially to be computable by most of code analysis tools. We do not deny the need of providing specific and so more powerful answers to some aspects of the component world such as the measure of granularity, the black-box vision and the concept of interface. But in the absence of dedicated standards and tools, we wanted to test the hypothesis that it is possible and appropriate to use some well-known object-oriented metrics to evaluate the quality of software components developed upon object-oriented technology.

To test our hypothesis we have conducted an experimental study on several OSGi applications. We have chosen a set of object-oriented candidate metrics that seems semantically consistent with the three levels of component-based development: internal, interface and application. Using descriptive statistics we have studied the distribution of each of theses candidate metrics among 10 OSGi applications. We have shown that these metrics can provide valuable information both at the component level (for developers) and at the application level (for architects). Their analyse allows to draw qualitatively and quantitatively what are a "typical" component and component-based application in the real world. This gives us the opportunity to discuss on the importance and the respect by the OSGi developers of some of good development practices pointed out by the literature. Finally, we have highlighted the ability of some of these metrics to predict two particular external quality measures: the number of revisions and the number of bugs. Our study has shown that some of the object-oriented metrics can be used by developers and architects to check the quality of component-based applications.

3.4 Expliciting Architecture Non-Functional Properties Using Architectural Patterns

Keywords: Architectural Patterns, Non-Functional Properties, Architectural Decisions,

Non-Regression Checking.

Participants: Minh Tu Ton That, Salah Sadou, Flavio Oquendo, Isabelle Borne.

The activity of architecting software systems can benefit from the concept of pattern, therefore providing a general reusable solution to a commonly occurring problem within a given context. It is the case of architectural patterns, a concept that enables to define a group of cohesive elements of a software architecture to solve a particular problem.

However, the current use of architectural patterns has a major shortcoming that needs to be addressed to leverage their use in complex software architectures. More specifically, in real world architectures recurring problems are complex and their solutions can be represented by patterns in complex forms that require the combination and reuse of other existing architectural patterns. Moreover, in a well defined context (a given company) there may be some recurring problems that no classical architectural pattern fits. Thus, two problems emerge: the definition of specific patterns and the construction of patterns by combining existing ones. In the literature, current support for pattern composition consists in fact of using merging operators that are not part of the pattern language. This limitation prevents the traceability as well as the reconstructability of patterns which are essential to solve for software evolution.

For addressing these open issues, we proposed an architectural pattern description language, called COMLAN, that has in particular the following properties: (i) it deserves first-class citizenship for both patterns and merging operators; (ii) it supports the design of hierarchical patterns. This language is graphical and can be easily integrated to an Architectural Description Language (ADL) environment.

With this language we propose the process of constructing patterns including two steps. The first step consists in describing a pattern as a composition graph of unit patterns. Thus, the pattern comprises many blocks, each block represents a unit pattern, all linked together by merging operators.

The second step consists in refining the composed pattern in the previous step by concretizing the merging operators. More specifically, depending on the type of merging operator, a new element is added to the composed pattern or two existing elements are mixed together. On the purpose of automating the process of pattern refinement, we used the Model Driven Architecture (MDA). Each pattern is considered as a model conforming to its meta-model in order to create a systematic process thanks to model transformation techniques. Thus, each refined pattern is attached to a corresponding pattern model from step 1 and any modification must be done only on the latter at step 1. At this stage, we offer the architect a pattern description language based on the use of classical architectural elements, architectural patterns and pattern merging operators.

We can see that through the two-step process, described above, any time we want to trace back the constituent patterns of a composed pattern in the second step, we can find them in its corresponding pattern model. Thus, we solve the traceability problem pointed out above.

We solve the second problem (reusability of merging operators) by the fact that merging operators are first-class entities in our pattern description language. In other words, merging operators are treated as elements of the pattern language where we can manipulate and store them in the pattern model like other elements. Therefore, the composition of patterns is not an

ad-hoc operation but a part of pattern. This proposal facilitates significantly the propagation of changes in constituent patterns to the composed pattern. Indeed, the latter can thoroughly be rebuilt thanks to the stored merging operators. So, merging operators not only do their job which performs a merge on two patterns but also contain information about the composition process. Thus, we think documenting them is one important task that architects should take into consideration.

Finally, to solve the third problem (support for hierarchical pattern composition), we propose to give pattern itself first-class status in our pattern description language. That means that patterns should play the same role as other elements where we can make connection with, add properties and most importantly, set them up as internal elements. This recursive definition of pattern gives the pattern description language the capacity to describe hierarchical patterns.

3.5 From Object-Oriented application to Component-based Application

Keywords: Component-based Architectures, Legacy Systems, Restructuring, Meta-Heuristic Approach..

Participants: Abderrahmane Seriai, Salah Sadou, Houari Sahraoui (Univ. of Montreal).

Nowadays, components are most often built using object-oriented technology. This is normal as, usually, new programming paradigms are defined using previous ones. That was the case for classes that are built on the procedural paradigm.

Classes are designed to hide their implementation (procedural details), i.e. encapsulation principle. This is also true for components that need to hide their implementation (details of contained classes). Components must expose only their provided and required interfaces. The organization of these interfaces should not reflect the implementation details of their component. They must be organized according to the component's business logic independently of classes that actually provide services exposed by the interfaces.

This problem appears when building components from scratch. However, it is more blatant during automatic restructuring of an object-oriented application into a component-based application. Indeed, in this case the approach of extraction of components is mainly based on the consistency of the groups of classes. In addition, the obtained components often have a large number of classes, which complicates the identification of their interfaces. There are two advantages of this type of restructuring: i) allowing a better understanding of the system to achieve its current maintenance thanks to its extracted architectural representation based on the component paradigm. ii) facilitating its future maintenance by its reimplementation within a component technology while relying on its architectural representation. Most of the work on the extraction of components from object-oriented applications consider components as clusters of classes with a set of provided methods and a set of required methods.

To the best of our knowledge, our past work is the lonely one that projects the extracted components on a concrete component model. In this work, the organization of the interfaces was achieved through a direct mapping to the component's classes. Although this projection is an improvement with respect to the existing work, it still reveals the details of the component implementation. The disadvantage is that if the definition of component interfaces

is strongly influenced by structural aspects, coming from the object approach (component's internal classes), it complicates the understanding of the relationships between components. Indeed, component interfaces should represent its functional aspects and its relationships with the other components should imply only that.

We propose to consider the problem of organizing the interfaces of a component starting from its classes and all their exposed methods. Thus, our starting point is the result obtained by the majority of the works on component extraction from object-oriented applications. More concretely, we view interface organization as a clustering problem based on dependencies between the exposed methods (services) and components that use them. Formal-concept analysis techniques are used to perform the clustering. The idea behind this is that when a component uses another, it means that it needs one or more specific aspects from the latter. The analysis of all interactions of a component allows us to define its various exposed aspects. Each aspect is represented by a subset of the exposed methods and will be implemented as an interface of the component. This applies to the provided interfaces. Similarly, in a context where components are extracted from the same application, the required interfaces of a component are defined according to interfaces provided by the other components that it uses.

Given the advantages of the component paradigm, re-engineering object-oriented applications into component-oriented applications seems to be a promising choice. It helps companies reducing their software maintenance and evolution costs. Especially if this process is automated in a large part. In this context, automated identification of components and their interfaces is an important first step. In this work, we have proposed an interface identification approach that is defined as a continuation of a component identification process. The aim is to build a consistent and understandable architecture that facilitates the maintenance and/or the mapping of the application to a concrete a component model.

4 Software

4.1 SoSmart Architect Studio

Participants: Flavio Oquendo, Isabelle Borne, Jérémy Buisson, Régis Fleurquin, Elena Leroux, Salah Sadou, Gersan Moguérou..

SoSmart is a novel environment for description, analysis, simulation, and compilation/execution of SoS architectures. These SoS architectures are described using the SoSmart language, which is a language based on process algebra, and on a meta-model defining SoS concepts. Because constituents of an SoS are not known at construction 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.

At the end of 2013, the meta-model has been defined, and the language is being heavily discussed.

4.2 Coqcots & Pycots

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. Then Pycots is the corresponding framework for the Python language.

Instead of the classical start/stop approach, Coqcots & Pycots foster an approach based on Dynamic Software Updating (DSU) to bring components to a behaviour suitable for dynamic reconfiguration. To this end, Pycots relies on Pymoult, a library developed at Télécom Bretagne, which provides many DSU mechanisms in a single platform.

4.3 COMLAN

Participants: Minh Tu Ton That, Salah Sadou, Flavio Oquendo..

COMLAN is a tool to design architectural patterns with the focus on documenting composition operations. It provides the following functionalities:

- Create architectural patterns
- Compose patterns using merging operators
- Refine the composed pattern

5 Contracts and Grants with Industry

5.1 Grants Involving Industry

Collaboration on systems for processing big-data (CIFRE)
Participants: MGDIS.

Collaboration on description of systems-of-systems (ARED)
Participants: DCNS.

Collaboration on analysis of systems-of-systems (CIFRE)
Participants: SEGULA.

5.2 National Cooperative Projects

PEPS API (Projet Exploratoire Pluridisciplinaire CNRS sur l'Automatique pour l'informatique autonomique)

- Funding: CNRS
- Period: July 2011 - July 2013
- Partners: LIG, INRIA Rhône-Alpes, GIPSA-Lab, Lab-STICC

- Objective: To explore how control theory, software adaptability and reconfigurable architectures contribute to each other in order to build autonomic systems.

5.3 International Cooperative Projects

AD-AUTO (Adaptive Software Architectures for Autonomic Systems)

- Funding: CNPq (Brazilian National Research Agency)
- Period: 2011 - 2013
- Partners: UFRN - Federal University of Rio Grande do Norte, UFPE - Federal University of Pernambuco (Brazil)
- Objective: To develop mechanisms for dynamic reconfiguration based on reflective architectural models and middleware supporting the architecture and engineering of autonomic systems.

ProSA-RAES (Framework to Support Reference Architectures for Embedded Systems)

- Funding: FAPESP (Sao Paulo State Research Agency)
- Period: 2012-2014
- Partners: Fraunhofer-Institut für Experimentelles Software Engineering IESE (Germany), ICMC Research Institute at USP (Brazil)
- Objective: To develop a formal language and framework for defining reference architectures of embedded systems based on ArchWare languages.

6 Other Grants and Activities

6.1 International Collaborations

- Salah Sadou:
  - Université de Montréal (Houari Sahraoui): Restructuring OO systems into Component-Based Systems. (PhD in co-tutelle)
  - Université des Sciences et de la Technologie Houari Boumedienne, Alger (Mohamed Ahmed Nacer): SoS Product Line Architecture. (PhD in co-tutelle)

6.2 National Collaborations

- Jérémy Buisson is an external collaborator of Télécom Bretagne (PASS team), contributing to the supervision of PhD student Sébastien Martinez of that team
- Salah Sadou has a collaboration on Reuse of Architectural Constraints with the Université de Montpellier 2 (Chouki Tibermancine and Christophe Dony)
- Flavio Oquendo has participated in a collective book on software architecture organized by the LINA - Université de Nantes (Mourad Oussalah).

7 Dissemination

7.1 Editorial Boards and Guest Editions

- Flavio Oquendo:
  - Springer Journal of Software Engineering Research and Development (Member of the Editorial Board)
  - Special Issue on Software Architecture Modelling of the Journal on Software and System Modelling (Guest Editor)
  - Special Issue on Software Components, Architectures and Reuse of the International Journal of Universal Computer Science (Guest Editor)
  - Special Issue Advanced Architectures for the Future Generation of Software-Intensive Systems of the Elsevier Journal on Future Generation Computer Systems - FGCS (Guest Editor)
  - International Journal of Artificial Intelligence and Applications for Smart Devices (Member of the Editorial Board)
- Régis Fleurquin:
  - Special Issue Advanced Architectures for the Future Generation of Software-Intensive Systems of the Elsevier Journal on Future Generation Computer Systems - FGCS (Member of the Guest Editorial Board)

7.2 General Chairs, Steering Committees

- Flavio Oquendo:
  - European Conference on Software Architecture - ECSA 2013 (Conference Chair)
  - IEEE/IFIP Working International Conference on Software Architecture - WICSA (Steering Committee Member)
  - European Conference on Software Architecture - ECSA (Steering Committee Chair)
  - Conférence francophone sur les architectures logicielles - CAL (Steering Committee Member)
  - IEEE International Conference on Collaboration Technologies and Infrastructures - WETICE (Steering Committee Member)

7.3 Program Chairs, Tutorial Chairs, Special Session Chairs

- Isabelle Borne: Research Project Symposium at ECOOP/ECMFA/ECSA conferences, Montpellier (Symposium Chair)
- Flavio Oquendo: International Workshop on Software Engineering for Systems-of-Systems - SESOS 2013 (Program Co-chair)

7.4 Program Committees

- Isabelle Borne:
  - CIEL: Conférence en Ingénierie du Logiciel, 2013
  - GPL: Journées du GDR GPL, 2013
  - SESOS: International Workshop on Software Engineering for Systems-of-Systems, 2013
- Jérémy Buisson:
  - ICCS: International Conference on Computational Science, 2013
- Régis Fleurquin
  - ECSA: European Conference on Software Architecture, 2013
  - TSI Journal 'L'objet": served as reviewer in 2013
- Flavio Oquendo:
  - ECSA: European Conference on Software Architecture, 2013
  - WICSA: IEEE/IFIP Working International Conference on Software Architecture, 2014
  - ISARCS: International ACM Symposium on Architecting Critical Systems, 2013
  - ICSEA: International Conference on Software Engineering Advances, 2013
  - ICSOFT-EA: International Conference on Software Engineering and Applications, 2013
  - WETICE: IEEE International Conference on Collaboration Technologies and Infrastructures, 2013
  - AROSA: IEEE WETICE Conference Track on Adaptive and Reconfigurable Service-oriented and component-based Applications and Architectures, 2013
  - PESOS: International Workshop on Principles of Engineering Service-Oriented Systems at ACM/IEEE International Conference on Software Engineering (ICSE), 2013
  - ICAART: International Conference on Agents and Artificial Intelligence, 2013
  - ICSOFT-PT: International Conference on Software Paradigm Trends, 2013
  - SBES: National Symposium on Software Engineering, 2013
  - SBCARS: National Symposium on Software Components, Architectures and Reuse, 2013
  - I-ESA: International Conference on Interoperability for Enterprise Systems and Applications, 2013
  - COMPUTATION: International Conference on Computational Logics, Algebras, Programming, Tools, and Benchmarking, 2013

  - ADAPTIVE: International Conference on Adaptive and Self-Adaptive Systems and Applications, 2013
  - ADVCOMP: International Conference on Advanced Engineering Computing and Applications in Sciences, 2013
  - WEA: International Workshop on Software Ecosystem Architectures, 2013
  - AAMAS: International Conference on Autonomous Agents and Multiagent Systems, 2013

- Salah Sadou:

  - CBSE: International ACM SigSoft Symposium on Component Based Software Engineering, 2013
  - CAL: Conférence sur les Architectures Logicielles, 2013
  - QUORS: IEEE International Workshop on Quality Oriented Reuse of Software, 2013
  - SESOS: International Workshop on Software Engineering for Systems-of-Systems, 2013
  - IET Software Journal: served as reviewer in 2013
  - Information and Software Technology (IST) Journal, Elsevier: served as reviewer in 2013

7.5 Invited Keynote Speakers, Invited Speaker

- Flavio Oquendo:

  - SBSI 2013: Systems-of-Systems: A New Frontier for Information Systems in an Open World (Invited Keynote Speaker)
  - CSBC 2013: Software-intensive Systems-of-Systems: Emerging Paradigm for Smarter Cities (Invited Keynote Speaker)

7.6 Conference Panels

- Flavio Oquendo:

  - SBSI 2013: IS Challenges in an Open World (Invited Panelist)

7.7 Thematic Schools

- Salah Sadou:

  - Maintenance and Software Evolution, Winter School of Université d'Oran, Algeria, on December 2013Â (Invited Speaker)

7.8 National Event Organization

- Salah Sadou:
  - Journée RIMEL (GDR GPL), 29 November 2013, Annecy

7.9 Doctoral Examination Boards

- Isabelle Borne: Chair of the Doctoral Examination Board of Ali Ghaddar (Une contribution à la gestion des applications SaaS mutualisées dans le cloud : approche par externalisation), Université de Nantes, 18 July 2013
- Isabelle Borne: Member of the Doctoral Examination Board of Abdoulkader Osman Guedi (Evolution de transformation automatique de modèles de systèmes d'information : une approche guidée par l'analyse formelle de concepts et l'analyse relationnelle de concepts), Université de Montpellier 2, 10 July 2013
- Flavio Oquendo: Reviewer in the Doctoral Examination Board of Mohamed Nadhmi Miladi (Une approche par transformation de modèles pour le déploiement et la gestion des architectures logicielles distribuées), Universités de Toulouse et Sfax, 2013

7.10 Scientific Networks

- Isabelle Borne: Co-chair of Action IDM - GDR GPL & ASR
- Salah Sadou: Co-chair of GT RIMEL - GDR GPL

7.11 Industrial Collaborations

7.12 Visiting Positions

- Flavio Oquendo: University of Sao Paulo, Brazil (Invited Professor, 2 months)

7.13 Research Excellence Awards (PES)

- Flavio Oquendo: PES A (2011-2015)
- Salah Sadou: PES A (2010-2014)

7.14 Teaching Responsabilities

- Régis Fleurquin: Head of Computing Department of IUT (UBS, Vannes), until September 2013
- Flavio Oquendo: Head of Computing Degree of ENSIBS School of Engineering, 2013
- Flavio Oquendo: Member of the Steering Board of ENSIBS School of Engineering, 2013

7.15 Expertises

- Salah Sadou: Evaluation of proposals submitted to the STIC AmSud Program
- Flavio Oquendo: Expert acting as reviewer and evaluator of R&D Projects for the European Commission (Framework Program 7) for:
  - Software-intensive Systems Engineering,
  - Systems-of-Systems, and
  - Trustworthy ICT.

7.16 National Council of Universities (CNU)

- Isabelle Borne: Member of CNU 27

8 Bibliography

Books and Monographs

[1] M. Ali Babar, I. Gorton, F. Oquendo, *SoSyM Special Issue on Software Architecture*, Springer, 2013, http://hal.inria.fr/hal-00913477.

[2] E. Almeida, F. Oquendo, *J.UCS Special Issue on Software Components, Architectures and Reuse: Modeling, Customization and Evaluation*, J.UCS, 2013, http://hal.inria.fr/hal-00913467.

[3] F. Oquendo, P. Avgeriou, C. Cuesta, J. C. Maldonado, E. Nakagawa, K. Drira, A. Zisman, *Software Engineering for Systems-of-Systems: Proceedings of the ACM Sigsoft/Sigplan International Workshop SESoS'2013, Montpellier, France*, ACM, 2013, http://hal.inria.fr/hal-00913491.

Articles in referred journals and book chapters

[4] M. Ali Babar, I. Gorton, F. Oquendo, "Building European Software Architecture Community: How far have we come?", *Journal on Software and System Modeling 12*, 2, 2013, p. 435–438, http://hal.inria.fr/hal-00913480.

[5] E. Leroux, F. Oquendo, Q. Xiong, "Test de conformité basé sur l'architecture logicielle", *Revue des Nouvelles Technologies de l'information (RNTI)*, December 2013, p. 01–18, http://hal.inria.fr/hal-00908580.

[6] E. Nakagawa, F. Oquendo, J. C. Maldonado, "Architectures de référence : concepts et processus", *in : Architectures logicielles : principes, techniques et outils*, Hermès Sciences, December 2013, p. 1–34, http://hal.inria.fr/hal-00913503.

[7] E. Nakagawa, F. Oquendo, J. C. Maldonado, "Reference Architectures", *in : Software Architecture: Principles, Techniques, and Tool*, Wiley, December 2013, p. 101–122, http://hal.inria.fr/hal-00913505.

Publications in Conferences and Workshops

[8] T. Gherbi, I. Borne, D. Meslati, "Towards an MDE Methodology to Develop Multi-Agents Systems Including Mobile Agents", *in: 8th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2013)*, SciTePress, p. 45–55, Angers, France, July 2013, http://hal.inria.fr/hal-00908558.

[9] S. Hamza, S. Sadou, R. Fleurquin, "Measuring Qualities for OSGi Component-Based Applications", *in: 13th International Conference on Quality Software*, p. 25–34, France, 2013, http://hal.inria.fr/hal-00912086.

[10] J. Leite, F. Oquendo, T. Batista, "SysADL: A SysML Profile for Software Architecture Description", *in: 7th European Conference on Software Architecture (ECSA 2013)*, *LNCS*, Springer, p. 106–113, Montpellier, France, July 2013, http://hal.inria.fr/hal-00913494.

[11] E. Leroux, F. Oquendo, Q. Xiong, "Architecture-Based Conformance Testing", *in: The Eighth International Conference on Software Engineering Advances (ICSEA'13)*, p. 55–64, Venice, Italy, October 2013, http://hal.inria.fr/hal-00874942.

[12] E. Leroux, F. Oquendo, Q. Xiong, "Test de conformité basé sur l'architecture logicielle", *in: 7ème Conférence francophone sur les architectures logicielles (CAL'2013)*, p. 01–14, Toulouse, France, May 2013, http://hal.inria.fr/hal-00875283.

[13] S. Martinez, F. Dagnat, J. Buisson, "Prototyping DSU techniques using Python", *in: HotSWUp'13*, USENIX (editor), p. https://www.usenix.org/conference/hotswup13/prototyping–dsu–techniques–using–python, San José, United States, June 2013, http://hal.inria.fr/hal-00907744.

[14] E. Nakagawa, M. Gonçalves, M. Guessi, L. Oliveira, F. Oquendo, "The State-of-the-Art and Future Perspectives in Systems-of-Systems Software Architectures", *in: First International Workshop on Software Engineering for Systems-of-Systems (SESoS 2013)*, p. 13–20, France, 2013, http://hal.inria.fr/hal-00913493.

[15] E. Nakagawa, F. Oquendo, "Perspectives and Challenges of Reference Architectures in Multi Software Product Line", *in: 1st International Workshop on Multi Product Line Engineering (MultiPLE 2013)*, ACM (editor), ACM, p. 100–103, Tokyo, Japan, August 2013, http://hal.inria.fr/hal-00913501.

[16] F. Oquendo, "Software-intensive Systems-of-Systems: Emerging Paradigm for Smarter Cities", *in: 33rd Congress of SBC (CSBC 2013)*, Maceio, Brazil, July 2013, http://hal.inria.fr/hal-00913512.

[17] F. Oquendo, "Systems-of-Systems: A New Frontier for Information Systems in an Open World", *in: 9th Conference on Information Systems (SBSI 2013)*, Joao Pessoa, Brazil, May 2013, http://hal.inria.fr/hal-00913506.

[18] R. Silva, V. Fragal, E. Oliveira-Junior, I. Gimenes, F. Oquendo, "SyMPLES: A SysML-based Approach for Developing Embedded Systems Software Product Lines", *in: 15th International Conference on Enterprise Information Systems (ICEIS 2013)*, p. 257–264, Angers, France, July 2013, http://hal.inria.fr/hal-00913498.

[19] T. M. Ton That, S. Sadou, F. Oquendo, I. Borne, "Composition-centered architectural pattern description language", *in: 7th European Conference on Software Architecture - ECSA, 7957*, p. 1–16, Montpellier, France, 2013, http://hal.inria.fr/hal-00912032.

Miscellaneous

[20] M. Ali Babar, I. Gorton, F. Oquendo, "Guest Editorial of the SoSyM Special Issue on Software Architecture", 2013, Software and System Modeling, Vol. 12, No. 2, http://hal.inria.fr/hal-00913483.

[21] E. Almeida, F. Oquendo, "Guest Editorial of the J.UCS Special Issue on Software Components, Architectures and Reuse: Modeling, Customization and Evaluation", 2013, Journal of Universal Computer Science, Vol. 19, No. 2, http://hal.inria.fr/hal-00913488.