Approach Based Patterns for System-of-Systems Reconfiguration

Franck Petitdemange
IRISA
University of South Brittany
Vannes, France
franck.petitdemange@irisa.fr

Isabelle Borne
IRISA
University of South Brittany
Vannes, France
isabelle.borne@irisa.fr

Jeremy Buisson
IRISA
Military Academy of St-Cyr
Vannes, France
jeremy.buisson@irisa.fr

***Abstract*—Systems-of-systems (SoS) are a particular class of systems that recruit dynamically their constituents to achieve a global goal. To accomodate this approach, the architecture of SoS is usually described by architectural patterns to be instantiated at runtime. Based on the study of an example, we introduce reconfiguration patterns to help reasoning on reconfiguration and maintaining the architectural patterns of the SoS.**

I. Introduction

System-of-systems (SoS) are a particular class of systems. An SoS is a set of Constituent Systems(CSs) which collaborate to fulfill a mission. Each CS has its own managerial independance, meaning that it has its own teams of managers, developers, administrators, maintainers who control its lifecycle (*e.g.*, deciding when the CS is started, stopped, updated) independently of the SoS. Each CS has also operational independance, meaning that the CS can fulfilled valid missions if disassembled from overall system. In this context, the architect who designs an SoS has autority on interactions between CSs but has no control over the CSs themselves. Due to the lack of coercitive power on the environnement of the SoS some architectural details are delayed until runtime. For instance a CS can leave or join the SoS on its own right, therefore the SoS must reconfigure its architecture at runtime to accomodate such changes.

The architecture of the SoS defines the role of each constituent system and the interactions between them. In order to be assisted with regard to this point, the SoS architect can use patterns. A pattern is a generic building block which defines a structure and behaviors as a solution to solve a well-identified design problem in a specific context with well-known properties.

One challenge of SoS engineering is that the constituent systems, because of their own managerial and operational independence, can decide to engage or disengage at their will. But regardless of the decisions of the constituent systems, the SoS must self-reconfigure to continuously achieve its global mission. In the case of SoS, we identify two cases for reconfiguration: either maintaining the SoS pattern or switching to a new coalition.

A difficulty arises because the role a constituent system plays after reconfiguration depends not only on its role before reconfiguration, but also on its involvement in other SoS as well as its own missions. In this paper we consider the example of a fire emergency system. Clearly, if an ambulance system is transporting injured persons at the time of the reconfiguration, this ambulance will not play the same role as if it were idle. To solve this issue, we propose to assist reconfiguration with patterns. Our idea is that the architect defines configurations and reconfigurations with patterns. Then reconfiguration patterns can be used as patches on the architecture in order to change it at runtime while maintaining the consistency of SoS and preserving the other patterns (*i.e.*, the other SoS).

Section II presents the main concepts of the SoSADL architecture description language we are relying on, as well as an illustrative example expressed in terms of these SoSADL concepts. Section III presents how we propose to use patterns to assist reconfiguration. Section IV provides an overview of related works. Section V concludes the paper with future directions.

II. Background

A. SoSADL Concepts

This paper focuses on the reconfiguration concern in SoS. To make the examples more concrete, we rely on SoSADL, a language under construction in our team targeted the SoS description architectures. SoSADL is an evolution of π-ADL [1], a language used to describe static and dynamic architectural specifications. π-ADL is based on the π-calculus, a formal basis to verify specification of architectural structures and behaviors. SoSADL extends π-ADL with new architectural artifacts for SoS. In comparison to other ADL, SoSADL allows to declare sets of architectural constraints solved at runtime to generate the configuration. This language relies on three main concepts.

- *Systems* are the constituents of an SoS. Following the definition of SoS, each system has its own mission and is operationally and managerially independent, while contributing to the global mission of the SoS. Systems are described in terms of their interaction points, called gates, and a model of their interaction behavior. The gates are the means given to the SoS to interact with the system such that the system achieves some tasks for the SoS.
- *Mediators* are communicating elements that specify the interactions among systems. Mediators belong to the

SoS, i.e., the SoS has full control over them. They are described in terms of their interaction points, which we call duties, and their internal behavior.
- Each SoS is described by a *coalition*, which identifies systems and controls mediators. Coalition are described by-intention. It means that each coalition is defined by 1) sets of to-be-discovered systems and sets of to-be-instantiated mediators; and 2) a constraint that describes how the system gates and mediator duties are bound to one another to establish communications. With SoSADL, we consider that SoS can themselves be systems involved in other SoS. Hence a coalition may expose some gates to its environment, like any other system. The resolution of the coalition constraint determines the instances of mediators that are required as well as the bindings between system gates and mediator duties. When systems are discovered or disengage from the SoS, the resolution of the constraint yields to a new architecture and triggers a reconfiguration.

B. Illustrative Example

To illustrate our proposal we study the case of an emergency organization. It is a reference case study commonly used in the SoS area [2], [3]. In this paper we focus the example on a single fire emergency service, independently of the other services that may be involed. This service is composed of several *ambulances*, *fire cars* and *officers* that are dispatched to emergency scenes by a *strategic command center* (SCC) which fulfills the global mission by controling the allocation of these elements.

When an incident scene requires the coordination of multiple resources (ambulances, fire cars and/or officers), the SCC creates a temporary *tactical command* (TC). An officer is in charge of the command and control of the resources allocated to the TC. Following a hierarchical command scheme, the TC issues operational orders to its allocated ressources based on strategic decisions received from the SCC. In this case, resources are subordinates of the TC, rather than the SCC.

The fire emergency service can be considered an SoS because:

- ambulances and fire cars are distributed over several districts in order to minimise intervention time (geographical distribution);
- an ambulance can operate standalone to rescue one wounded (operational independence);
- in such situation, the ambulance has its own strategy to operate and control its own cycle life (managerial independence); and
- the number of ambulances and fire cars changes over the time (evolution).

The constituent systems (CSs) of the fire emergency service are SCC, ambulances, fire cars and officers. Because we consider a hierarchical command chain (coercive power on CSs), the SoS belongs to the *directed* category [4].

C. Modelization of the example in terms of SoSADL concepts

In our example, we identify two coalitions: the global architecture of the emergency service; and the adaptation of the service to specific incident scenes.

1) Fire emergency service: The fire emergency service contains four sets of constituent systems: one SCC, some officers, some ambulances and some fire cars. It contains a set of mediators named *transmitter* which model communication equipment. The coalition defines the SCC as the central authority that is responsible for the control and command of the other CSs. The coalition constraint states that a transmitter handles the communications between SCC and the CSs of a district: it broadcasts orders from SCC to CSs and transmits reports from CSs to SCC. As consequence, the SoS must instantiate a set of transmitters in order to comply to this rule. As shown in Figure 1, to resolve the coalition constraint, the SoS instantiates one transmitter when all the CSs are located in the same district. But in the case the ambulances, officers and fire cars are dispacthed over several districts the SoS instantiates several transmitters.

2) Tactic command: When an incident occurs in the environment of the SoS, a tactical command coalition is temporarily instantiated by the SCC, resulting in a reconfiguration of the global architecture. The SCC reshapes the coalition, introducing a new intermediate authority (an officer) and allocates some resources to it. The new constraint of the reconfigured coalition states that communications must pass through the officer, involving a second set of transmitters between officers and SCC. SCC maintains direct authority on officers. The reconfiguration brings for instance the SoS from Figure 1 to Figure 2: transmitter 2 is instantiated and ambulance and fire car are connected to it. Once again, depending on geographical distribution of the CSs, several transmitter 1 and several transmitter 2 may be instantiated to accomodate hardware constraints of the equipment modeled by transmitters.

III. Patterns for Dynamic Reconfiguration

Architectural patterns provide a high level view of a system architecture and define rules to support software construction with well-defined properties. According to [5]:

> Architectural patterns capture important structure, practice, and technique that are key competencies in a given field.

Due to the increased complexity of SoS (compared to regular systems), the use of architectural patterns is mandatory to help architects understanding and documenting the behavior and structure of the SoS [6]. Existing projects such as COMPASS [7] and DANSE [8] study how existing patterns in the literature may be applied in a SoS architecture.

Constant evolution is a key feature of a SoS, but this topic is not fully addressed in classical architectural patterns. While, e.g., [9], [10] verify that primitive reconfiguration actions preserve the architectural invariants of their respective component model, the challenge for the SoS field arises from

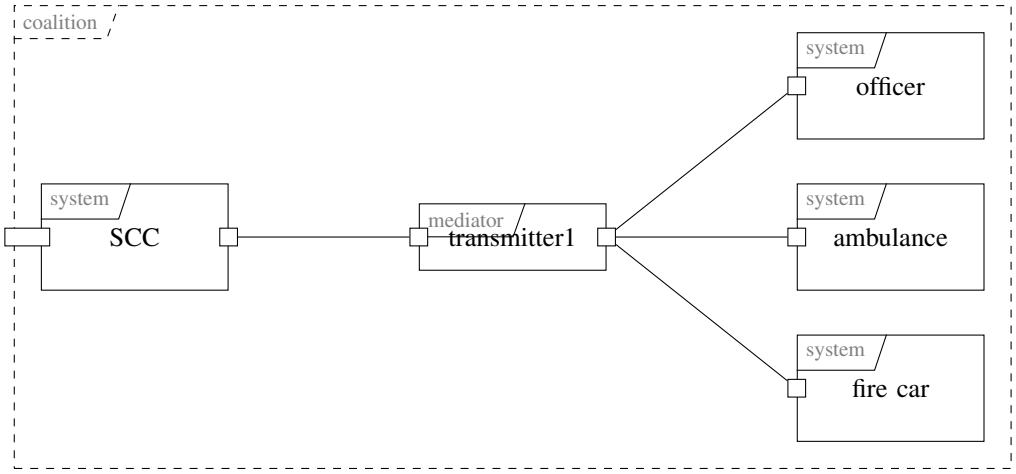

Fig. 1. Pattern of the fire emergency service coalition

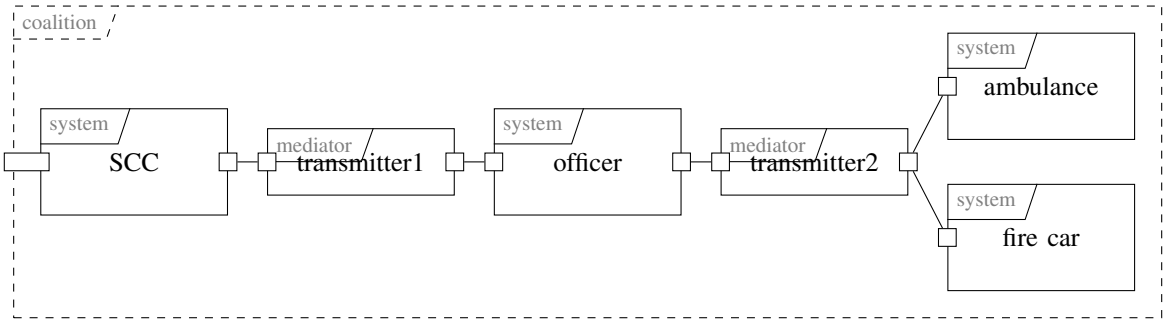

Fig. 2. Pattern of the fire tactical coalition

the generalized pattern-based new approach to architecture description. Like suggested by the constraint-based approach of SoSADL, the whole SoS architecture can be based on an architectural pattern. A dynamic reconfiguration must not compromise this architectural pattern used in the SoS.

The objective of this work is to explore a new approach in order to reason about reconfiguration with specific patterns.

The architect could apply a set of reconfigurations described by reconfiguration patterns. Reconfiguration patterns should preserve the architectural decisions defined through architectural patterns. In our work we focus on the relation between architectural patterns and reconfigurations.

We identify two cases for SoS reconfiguration:

- On the one hand, reconfigurations should maintain consistency in the hierarchy of architectural patterns defined in the SoS. The architectural patterns will be instantiated by coalitions, and re-instantiated when CSs leave or join. Based on our example (figure 1), if a CS loses its connection with SCC by moving to another district (out of range of its transmitter), a new transmitter must be instantiated for that district (if it does not exist) and then the CS reconnected to this transmitter.
- On the other hand, reconfigurations must allow the evolution of the hierarchy of patterns with composition and fusion operations of other patterns. At runtime when the SoS creates a new coalition, the process which selects and composes CSs must preserve the consistency in the SoS. For instance, when an incident occurs, the SCC may want to create a new tactical command coalition. When multiple incidents occur at the same time, several tactical commands could be instantiated. Consider that a tactical command assigns an ambulance to transport an injured person to the hospital: in this case the SCC cannot disengage this ambulance to provision a new tactical command. According to the domain knowledge and current state of SoS, dynamic reconfigurations should determine a set of reconfiguration operations that preserve consistency, and select and compose a new coalition.

The former clearly concerns the repeated resolution of the constraint in order to instantiate the pattern depending on the joining and leaving CSs. The latter shows that the role of the CSs depends not only on the target pattern, but also on their role before reconfiguration in regard to both the SoS and the environment.

The reconfiguration patterns will be designed by identifying the steps and constraints of a reconfiguration case. For example starting from the main coalition (figure 1) to get a tactical command coalition (figure 2) constitutes a reconfiguration

pattern composed of smaller patterns that explain the necessary steps.

The main coalition will be described with SoSADL and the reconfiguration patterns will include some SoSADL templates to guide the reconfiguration process.

Several advantages could be excepted. Relating architectural and reconfiguration patterns, we will describe what can be reconfigured in a SoS and how to achieve these kinds of reconfiguration. The combination of multiple concurrent reconfigurations should also be better controlled if the effects of well-designed reconfigration patterns are bounded by the related architectural pattern. Reconfiguration patterns will be a means to reason about reconfiguration in SoS. Using an adequate language, such as SoSADL, will allow to check and validate the reconfiguration operations. This approach should allow to have the traceability of the reconfiguration process.

IV. Related work

Some major European projects gave some ideas to deal with reconfiguration of SoS (COMPASS and DANSE european project). COMPASS addresses the problem of emergent property dependences against dynamicity (changes in SoS configuration) and evolution (changes in the behaviours of SoS). This dependability relies on a *contract* approach [11], to specify CSs behaviors, and a framework to model and reason about fault tolerance. This project also proposes a collection of architectural patterns [12] extracted from literature (software system and engineering) which adress particular needs of a SoS.

Another European project named DANSE [8] proposes an approach of development process for architectures based on architectural patterns. It demonstrates the usefulness of architectural patterns in SoS enginering (modeling, analysis, dealing with complexity). Architectural patterns are used to generate optimized architectures. From informations extracted from architectual patterns (e.g connections between CS, constraints on their behaviors and capacities) a solver generates an optimized architecture which satisfies goals of the SoS. The approach does not address issuing reconfiguration operations to switch to the target architecture from current one. To deal with dynamicity and evolution, DANSE has abstracted models from SoS characteristics and used a contract approach to define semantics on behaviors, evolutions and structural parts of SoS [13]. Our approach mainly addresses evolution of SoS with a dynamic reconfiguration process based on reconfiguration patterns. As such it is a complementary approach to those projects.

Regarding dynamic reconfiguration, [9], [10] address the preservation of invariants solely at the primitive operation level. Both focus on invariants coming from the component model, while considering that the approach generalizes well to invariants coming from patterns or from the application. In our work we aim to propose reconfiguration patterns similar to architectural patterns, i.e., templates to be instantiated for each application. Our approach is complementary to those previous work.

V. Future Work

In this paper we proposed an approach to describe a particular kind of SoS and how we intend to manage reconfigurations in a SoS architecture. We propose to assist reconfiguration with a pattern-based approach. Relying on patterns implemented by an SoS, our approach suggests a set of reconfiguration patterns which could be applied to maintain consistency in the SoS. As mentioned reconfiguration implementation should follow the reconfiguration pattern. Another case study in our team based on a real situation : a flood monitoring system developed at the University of São Paulo, will allow us to collect more reconfiguration examples. We are starting by decomposing the reconfiguration steps and defining the elementary patterns. We will study how to compose the elementary patterns [14] to form reconfiguration patterns. Finally we will design a catalog of architectural patterns devoted to SoS and will build a system of patterns, where each pattern will describe an architectural solution to a SoS reconfiguration problem.

References

[1] F. Oquendo, "pi-ADL: An Architecture Description Language Based on the Higher-order Typed pi-calculus for Specifying Dynamic and Mobile Software Architectures," *SIGSOFT Softw. Eng. Notes*, vol. 29, no. 3, pp. 1–14, 2004. [Online]. Available: http://doi.acm.org/10.1145/986710.986728

[2] M. Forcolin, P.-F. Petrucco, R. Previato, R. L. Stevens, R. Payne, C. Ingram, and Z. Andrew, "Accident response use case engineering analysis report using current methods & tools," COMPASS Project, Tech. Rep., 2013.

[3] T. Lochow, I. Sanduka, R. Bullinga, A. Arnold, R. Kalawsy, G. Cristau, M. Jung, C. Etzien, and E. Honour, "Concept alignment example description," DANSE Projet, Tech. Rep., 2013.

[4] W. M. Maier, "Architecting principles for systems-of-systems," *Systems Engineering*, vol. 1, no. 4, pp. 267–284, 1998.

[5] J. O. Coplien, *Software patterns*. New York; London: SIGS, 1996.

[6] R. S. Kalawsky, D. Joannou, Y. Tian, and A. Fayoumi, "Using architecture patterns to architect and analyze systems of systems," *Procedia Computer Science*, vol. 16, pp. 283–292, 2013.

[7] C. Ingram, R. Payne, J. Holt, F. O. Hansen, and L. D. Couto, "Modelling patterns for systems of systems architectures," COMPASS Project, Tech. Rep., 2013.

[8] R. Kalawsky, D. Joannou, A. Bhatt, K. Ramalingam, I. Sanduka, M. Masin, E. Shindin, and U.Shani, "Report on danse architectural approaches," DANSE project, Tech. Rep., 2014.

[9] M. Léger, T. Ledoux, and T. Coupaye, "Reliable dynamic reconfigurations in a reflective component model," in *Proceedings of the 13th International Conference on Component-Based Software Engineering*, ser. CBSE'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 74–92.

[10] J. Buisson, E. Calvacante, F. Dagnat, E. Leroux, and S. Martinez, "Coqcots & pycots: Non-stopping components for safe dynamic reconfiguration," in *Proceedings of the 17th International ACM Sigsoft Symposium on Component-based Software Engineering*, ser. CBSE '14. New York, NY, USA: ACM, 2014, pp. 85–90.

[11] W. ling Huang, J. Peleska, K. Pierce, and U. Schulze, "Contract support for evolving sos," COMPASS Project, Tech. Rep., 2014.

[12] S. Perry, J. Holt, R. Payne, J. Bryans, C. Ingram, A. Miyazawa, S. Hallerstede, L. D. Couto, A. K. Malmos, J. Iyoda, M. Cornelio, and J. Peleska, "Final report on sos architectural models," COMPASS Project, Tech. Rep., 2014.

[13] C. Etzien, T. Gezgin, R. Passerone, A. Arnold, L. Mangeruca, and E. Shindin, "Danse modelling formalism, including domain metamodel & semantics: Focused on support for analysis and optimization," DANSE project, Tech. Rep., 2014.

[14] M. T. T. That, S. Sadou, F. Oquendo, and I. Borne, "Composition-centered architectural pattern description language," in *Software Architecture - 7th European Conference, ECSA 2013, Montpellier, France, July 1-5, 2013. Proceedings*, 2013, pp. 1–16.