Assisting the evolutionary development of SoS with reconfiguration patterns

Franck Petitdemange, Isabelle Borne, Jeremy Buisson

Assisting the Evolutionary Development of SoS with Reconfiguration Patterns

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

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

Jérémy Buisson
IRISA Military Academy of St-Cyr
jeremy.buisson@irisa.fr

ABSTRACT

System of systems (SoS) engineering is an emerging approach to deal with complex systems that have low or no control over their constituents. An SoS must adapt itself not only to the willingness of its constituents to participate or disengage, but also to evolving needs. As reconfiguration is a routine task, the SoS architect needs specific assistance, especially to easily identify reusable solution principles and to track her/his decisions in the design of reconfiguration. To address these issues, we propose to introduce the concept of *reconfiguration pattern* based on prior design and architectural patterns. These patterns constitute well-documented, principled and adaptable solution building blocks. We illustrate our approach with one reconfiguration pattern applied on an example based on the French communication system for emergency services.

1. INTRODUCTION

Systems of systems [13] are an emerging class of complex systems characterized by the combination of the following properties: (1) operational independence of the constituents; (2) managerial independence of the constituents; (3) evolutionary development; (4) emergent behavior; (5) geographical distribution of the constituents. Orthogonal to the above characteristics, four categories are often used to classify the SoS depending on the type of their management. A *directed* SoS is built such that the SoS has coercive control over its constituents. In an*acknowledged* SoS the design of the constituents is based on a cooperative agreement between the SoS and the constituents. In a *collaborative* SoS, the constituents collaborate on their own will when they agree on a common global service (called *mission*) they should provide. A *virtual* SoS gathers constituents that collaborate regardless of any well-identified global service. Two characteristics are challenging. First, the *constituent systems* retain their own operational and managerial independence, therefore the system of systems has low or no control over its constituents. Second, the emergent behavior results in the fact that architect cannot anticipate the interaction results between constituents. Regarding sustainability aspects of SoS, the architect needs to continuously evaluate the current architecture regarding the requirements. If a SoS diverges from requirements, the architect needs to compute a new architecture and deploy it. This deployment comes at run-time mainly. SoS usually involve critical systems or economic systems (e.g emergency services, transportation services) than can not support the disruption or degradation of some services. The SoS architecture continuously evolves: it is what we call evolutionary development.

The SoS domain emphasizes the need for reconfiguration. In component based systems, components are considered with homogeneous reconfiguration semantics and a complete control on operational behavior and connections. For example in the Fractal component model[1], components are assumed to implement the same reconfiguration semantics and a component can be stopped independently of its execution state. In a SoS, the reconfiguration semantics between constituents is not uniform. Each constituent retains a particular reconfiguration semantics, and to stop an ongoing operation is not the rule. Alternatives must be designed for each need of reconfiguration at run-time. As a consequence we advocate that specification, design and validation for reconfiguration become time consuming, and that it would be useful to provide assistance to the architect.

To deal with SoS architecture complexity and to assist architects, various kinds of architectural patterns were proposed [11, 12, 15]. These patterns are suitable to assist the architect with development, maintenance and evolution steps. They are reusable units for specification, design and analysis of architecture. We propose to use the same abstraction to address reconfiguration issues. Our approach fosters the reuse of principled solutions and the documentation of the decisions made in the design of each SoS reconfiguration.

Our contribution in this paper is the design of reconfiguration patterns applied to system of systems architectures. In the literature, reconfiguration patterns do not consider SoS context. In [10] the reconfiguration patterns mainly concern components and is based on the quiescent state of components. The managerial independence of constituents on a SoS does not allow to control their states. [14] propose to describe basic reconfiguration operations. It does not address the complex interdependencies involved in the reconfiguration. This could lead to inconsistencies on other parts of the SoS.

ACM ISBN 978-1-4503-4781-5/16/11.
DOI: http://dx.doi.org/10.1145/2993412.3004845

[1] http://fractal.ow2.org/

In our proposition, reconfiguration patterns could be considered at various levels. Coarse-grained patterns describe overall and global reconfiguration strategies. Fine-grained patterns describe solutions to specific reconfiguration issues. We advocate that patterns with coarse-grained identify the technical issues that need to be solved by fine-grained patterns. They will capture the context of managerial independence. To illustrate our contribution we define a pattern and we apply it to an SoS example. We show how this pattern effectively helps the architect in the design of reconfigurations.

The paper is organized as follows. The next section further describes the systems of systems approach and its particularity with regard to reconfiguration. Section 3 presents the concept of reconfiguration pattern based on its documentation, reuse and composition aspects. Section 4 describes one concrete reconfiguration pattern, as well as an example of it use. Section 5 discusses related work. Finally we conclude in Section 6 and give future directions.

2. APPROACH

Due to the intrinsic nature of SoS, the design and implementation of reconfigurations are routine activities during SoS operation, hence calling for a more effective and streamlined engineering process. Proposing reconfiguration patterns aims at improving in this regard. When the patterns are well-documented, they provide principled solutions to the most frequent situations with respect to the capabilities and willingness of the constituents to contribute to the reconfiguration. When no pattern matches the situation, the architect is still able to design and implement an *ad hoc* reconfiguration. The case for SoS is not a fundamental shift in terms of dynamic reconfiguration, in that the basic mechanisms can be similar to those in other adaptive systems. But an architect who needs to reconfigure a system of systems faces several difficult tasks related to managerial independence and emergence. She/he needs to define a new architecture, but also to make design choices in accordance with the constraints imposed by the SoS constituents, in order to create a viable reconfiguration design, which can later be implemented. The *reconfiguration patterns* we propose aim at providing the architect with assistance in these tasks. Following the idea of design patterns, architectural patterns, etc., the reconfiguration patterns capture typical solutions to reconfiguration problems with well-known properties. Patterns help in reusing reconfiguration knowledge and expertise in order to streamline this activity.

2.1 Issues in the design of a SoS reconfiguration

The documenting aspect of the reconfiguration patterns follows from the architect's requirements expressed whenever a reconfiguration is considered. As usual, the architect has to identify and express the changes that occur within the architecture. The changes of a SoS architecture are made of additions and removals of constituents, as well as changes of connections between the constituents. Namely, Changes describe how to go from the initial architecture to the target architecture while preserving the coherence of the running system. The architect must preserve transaction mechanism during the reconfiguration, for instance, to ensure that any transaction started ends and there is no deadlock.

Moreover, the managerial independence of constituents in the SoS leads to consider that each constituent is designed, deployed and managed independently of the other constituents and of the SoS. Each constituent may expose specific or restricted reconfiguration capabilities. As a consequence the SoS architect needs to make the constituent reconfiguration model explicit to deal with constituents having heterogeneous capabilities. An explicit model with formal semantics helps to identify a situation in which applying a pattern. For instance, the STOP operation does not have the same semantics in a directed or a collaborative SoS: in the former case, a constituent totally stops its activity; in the latter case, a constituent stops its contribution to the SoS, but it possibly still operates in other contexts. The more control the SoS has over its constituents, the more cooperative the constituents are, including with respect to reconfiguration. Therefore, the solutions to reconfiguration issues must be clearly settled in the context of a well-identified type of SoS management.

Even if the SoS architect can influence the constituents management, a SoS can not forcibly instantiate, stop, start or use a given constituent. When the SoS delegates some of the reconfiguration operations to its constituents, the latter decide how and when to perform these operations. We introduced the mediators to manage interactions between constituents. Thus we need a well-documented behavioral model of the mediators.

Some architectural requirements can be translated into architectural invariants. For instance, a requested invariant can be: given two constituents, there is always one of them being wired in the architecture, *i.e.*, even during the reconfiguration, a service remains continuously provided by either one of the two constituents. The architect can further quantify the acceptable service degradation or minimal service preservation. Behavioral properties are also to be considered during the reconfiguration. For instance, a message order may or may not be preserved between two constituents; real-time deadlines may or may not be relaxed; failures may or may not be taken into consideration; and so on. Such behavioral properties may require specific reconfiguration actions such as the inclusion of temporary constituents in order to mitigate transient unavailability of some other constituents. This can have an impact on constituent implication in the SoS. Indeed, the operational independence characteristics results in SoS constituents having competing interests and priorities for their own services.

2.2 The reconfiguration pattern form

The description of a reconfiguration pattern is made of the following fields. Some of them are usual general fields found in any kind of pattern, such as the name, intent, related patterns, forces. Some other fields are specific to the reconfiguration domain, such as the explicit reconfiguration grammar, used to deal with managerial independence. The fields we propose are:

- *Name* meaningfully identifies the pattern in catalogs.
- *Intent* informally describes the changes that are addressed by the pattern. It gives an indication about when the pattern can be applied.
- *Context* indicates the situations whenever the pattern can be considered, and the associated vocabulary used within the pattern. As such, the context is further refined by two main aspects:

- *Related architectural pattern* optionally refers to a well-documented architectural context, including specific terminology, structural constraints and properties.
- *Category of SoS management* documents what kind of SoS is considered.

- *Problem* refines the intention with the objectives and goals of the pattern. It explains why the pattern addresses a non trivial situation. The problem field contains the following items:
  - *Initial/target architectures* describe the starting and the ending points of the reconfiguration addressed by the pattern. While any ADL can be used here, graph-grammar-based, *e.g.*, [6] or logic-based, *e.g.*, [1, 3, 16] languages allow to abstract over the unchanged parts of the architecture.
  - *Architectural invariant* and *Behavioral properties* explicit the conditions ensured by the pattern during reconfiguration. They can be given as an informal prose or formalized using, *e.g.*, standard [3, 16] or temporal logic [7, 14].
  - *Forces* explain the general principles and points adopted in the pattern solution. They motivate these principles with respect to the objectives depicted in the problem.

- *Solution* is the advocated means by which the problem can be solved. It may contain some artifacts like (template) reconfiguration scripts, for instance. It includes a *Reconfiguration grammar* that specifies the capabilities of the reconfiguration mechanisms assumed by the participants of the pattern. At first glance, a statechart is used to express the reconfiguration grammar. Formal logical approaches [3, 16] provide a more precise specification with unambiguous semantics.

- *Consequences* discuss impacts of the solution, especially in terms of quality attributes.

3. EXAMPLE OF A RECONFIGURATION PATTERN

To illustrate our approach, we first give the description of a reconfiguration pattern. Then, we discuss how this pattern is actually used to reconfigure a specific system of systems.

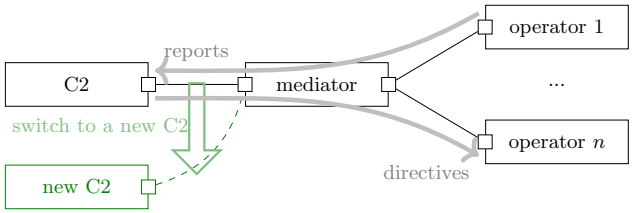
3.1 Co-evolution of configurations pattern

- Name: Co-evolution of configurations

- Intention: The co-evolution of configurations pattern allows to deploy a target architecture concurrently to a source architecture without stopping the running services. It is possible to change a service provider in a transparent way for consumers. Consumers progressively migrate from the source to the target architecture while states shared between source and target architectures co-evolve.

- Context:

Figure 1: Architecture diagram for the co-evolution pattern



  - *Related architectural pattern* This reconfiguration pattern is applied to the field of emergency SoS in which a constituent called *Command and Control (C2)* manages the other constituents called *operators* (cf. fig. 1). Emergency SoS (see fig. 4) usually contains two kinds of constituents. The *C2* (a single one) oversees the realization of the mission (SoS global service) and provides the state of the situation based on reports received from the other constituents. It coordinates the individual tasks of the other constituents, the *operators*, depending on its view of the mission (see Fig. 1). The operators achieve the tasks that are required to fulfill the mission assigned in the directives sent by the C2. They report their progress to the C2. Placed in the middle, a mediator observes and transmits the messages between the constituents.
  - *Category of SoS management* This pattern fits well to any directed or acknowledged SoS since it requires that the C2 provides its internal state for the purposes of the migration and a synchronization mechanism during reconfiguration. Moreover the SoS offers capabilities to observe and exchange control between systems. Despite the transient phase, the pattern ensures that the new C2 acquires a consistent view of the mission.

- Problem: The problem lies in the fact that the operators can hardly be reconnected atomically to the new C2, exactly at the same time that mission knowledge and supervision are transferred from the old C2 to the new one. Because of their own managerial independence, the operators freely choose when to disconnect from the old C2 and start reporting to the new one. Consequently, there is a transient phase during which some of them still report to the old C2 while the other ones have already switched to the new C2. If no specific action is taken, the two C2 have divergent views of the mission, and after the reconfiguration, the view of the new C2 is inconsistent. Due to this, the new C2 may order incorrect actions that can compromise the mission. Moreover, the SoS can not force the operators to change their connections in the SoS.
  - *Initial/target architectures* The figure (see fig. 1) shows the reconfiguration issue. The architect wants to replace the C2 while preserving the executed mission.
  - *Architectural invariant* and *Behavioral properties*

Figure 2: Behavior of the mediator in the co-evolution pattern

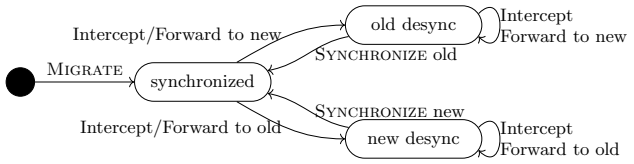
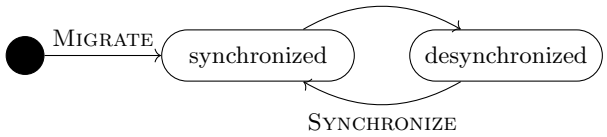


Figure 3: Reconfiguration grammar for the C2 systems in the co-evolution pattern



  - ∗ Behavioral property: the two C2 have the same view of the mission.
  - ∗ Structural invariant: at least one of the two C2 is connected.

  new C2 shares the same state of the mission
- *Forces* of the pattern are:
  - ∗ Maintaining the consistency of communications between the constituents.
  - ∗ Preserving the status of constituent collaborations in the SoS: until all the operators are connected to the new C2, the old C2 preserves its supervision capacity and knowledge on the mission.

- Solution: In order to not suspend the mission, the solution consists in concurrently deploy the initial and final architectures. The operators are migrated individually. During the transient phase, when the two C2 are contributing to the SoS, they are synchronized by means of scripts provided by the architect, in order to share the same view of the mission. The synchronization task is devoted to the mediator, such that the pattern does not have to assume any specific collaboration from the constituents.

  - *Reconfiguration grammar*

  The C2 constituent should provide the reconfiguration grammar depicted by Fig 3:

  - Migrate: transfers the internal state from the old C2 to the new one. In the old C2, Migrate means exporting the internal state; in the new C2, it means importing the internal state. Because we consider the case of directed and acknowledged SoS, the SoS architect has to ensure that the two C2 use compatible state representations.
  - Synchronize: propagates the changes made to the internal state from one C2 to the other one. As for the Migrate operation, the SoS architect is responsible to ensure that the two C2 use a compatible encoding.
  - The transition to the de-synchronized state does not result from any specific action. The C2 are unaware that their view of the mission is not consistent anymore. Only the mediator can observe the de-synchronized state.

  This grammar ensures that the C2 may exchange state during the reconfiguration. This allows the new C2 to be initialized with the mission state of the old C2.
  As depicted in figure 2 a mediator has the ability to intercept the messages forwarded to C2s. This behavior allows the two C2 to be continuously synchronized during the reconfiguration. The solution is decomposed in the following steps:

    - ∗ Before any message can reach the new C2, the mediator initiates a state migration from the old C2 to the new one. The messages to the C2 are postponed until migration is completed. After migration all C2 have been synchronized.
    - ∗ When the mediator intercepts a message to a synchronized C2, the other C2 is marked as *de-synchronized*, and the intercepted message is forwarded and delivered to its destination.
    - ∗ When the destination of an intercepted message is a desynchronized C2, the synchronization script is triggered before message delivery.

    This behavior is implemented by the mediator for the time of the reconfiguration.

  The reconfiguration completes when all the operators are connected to the new C2. The architect can provide an additional protocol in order to deal with the case where some operators are reluctant to connect to the new C2.

- Consequences:

  - Communication consistency is maintained. Operators preserve their connection to the old C2 until they have not migrated to the new C2
  - Availability is maintained. The SoS is still functional during reconfiguration.
  - Reliability is partially maintained, since information reported by the operators to the C2 are preserved. Nevertheless, the pattern does not deal with de-synchronized C2 sending contradictory or inconsistent directives.
  - Performance is affected depending on how many times synchronization scripts should to be called. The cost of the synchronization can be mitigated by the use of techniques such as concurrent transactions.

3.2 Reconfiguring with the help of reconfiguration patterns

To illustrate our approach we chose the case study of the National shared Infrastructure of Transmissions (INPT)[2], which is an information management SoS used by various French emergency services in case of accidents. This system is particularly used for the resolution of incidents involving interoperability of communications. In this context, the SoS belongs to the category of acknowledged SoS: a SoS-level coordinator (the Ministry of the Interior in our case) defines how each service shares its communication infrastructure and information even if the service remains independent.

Figure 4 is a possible instance of this architecture (white elements only). Even if this instance is kept simple, it is representative of the reality. The system constituents are:

- *CODIS (Operational center)* has the mission to coordinate and to supervise operational activities.
- *PC (Command Post)* appears to assist CODIS mission in big disasters. It manages operational activities for a particular disaster.
- *Relay* is a communication infrastructure shared between emergency services. Its mission is to relay and route information.
- *Resource (R)* is an emergency resource that operates in the SoS as humans or vehicles (fire-car, ambulance, police car). Each resource follows its own mission bound to its field of expertise. They receive a mission from the CODIS and communicate mission report by using the upCodis port.
- *Mediator* are mobile radios that define how PC, Resource and CODIS interact with Relay (communication protocol, data and interaction points).

Emergent behavior is defined by the capacity of constituents to reduce disasters. It is produced by the involvement of different constituents from different emergency services which are coordinated by the CODIS. Regarding the managerial independence each constituent is subordinated to the CODIS but retains its own managerial independence.

A reconfiguration case can come from the operational environment. For instance, the CODIS receives a call from citizens who notify a strange smell. The CODIS engages a new mission with several constituent systems from various emergency services. On the spot the constituents discover a chemical disaster with loss of lives, goods and assets. Additional resources are engaged and the CODIS must control and command a lot of resources. To support this overload the CODIS needs to delegate a part of the mission management to a PC, resulting in the new configuration shown in Figure 4. In the reconfiguration pattern presented in the previous section, CODIS and PC play the C2 role and Relay is an operator.

The architecture evolution involves that the CODIS delegates its coordination mission to a PC, and that resources delegated to the mission, switch to the PC channel. Particular protocols are designed to allow commandment delegation in the SoS. The CODIS can share its channel security key with the new commandment in order to talk with the resource. The CODIS can allow the PC to access to its service mission.

To preserve the emergence, the architect has to maintain the coordination of services available during the reconfiguration. He/she must identify what problems need to be solved for ensuring this property (e.g does a deadlock can occur? is the coordination service still reliable?). Then, the architect must produce a specification in response to her/his analysis. Finally the architect designs a reconfiguration in accordance with the specification. The process of realization of the reconfiguration is guided by the pattern.

- First, the architect has to identify a reconfiguration strategy that matches with her/his intention (coarse-grained pattern). The co-evolution pattern can be selected.
- Next, the architect needs to check if the context is similar to his/her architecture. Indeed, the architecture pattern captures system inter-dependence. If the architect is not an expert in reconfiguration problem, the pattern provides explanations in the problem part.
- In the next step the architect must refine the reconfiguration strategy with the technical points identified in the reconfiguration pattern : suspend communication, synchronization script. We assume that the current architecture provides all the conditions to apply the solution. If it not the case, this implies that either the reconfiguration pattern must be composed with another reconfiguration, or the architect must choose another pattern. Then, the architect produces a specification by taking into account invariants (explained in the problem section) that is the mission of C2 is synchronized during the reconfiguration. Finally, the architect follows the solution to obtain his/her reconfiguration.

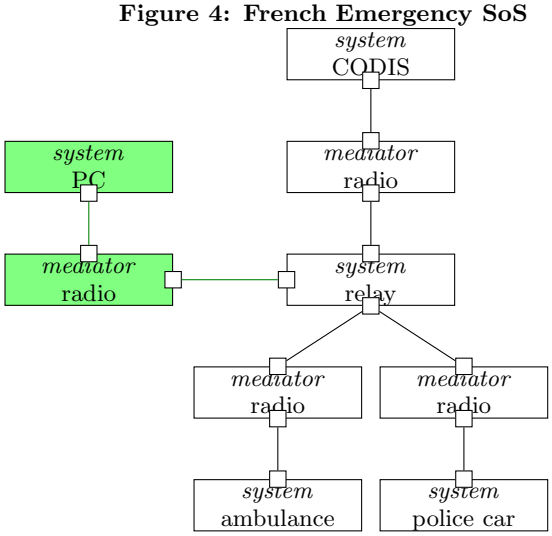
To resume, the reconfiguration is a complex engineering task. Indeed the architect can not impose a new configuration to systems and the SoS mission is critical and requires a safe reconfiguration. The pattern approach assists the architect in the different reconfiguration steps: specification, design.

4. RELATED WORK

The COMPASS and DANSE projects are the two major EU-funded projects about the system-of-systems methodology for the 2010-2015 period. Only the DANSE project studied the reconfiguration problem. In this project, the proposed methodology [19] consists in renewing the needs assigned to the system of systems whenever its behavior differs from prediction. Based on architectural patterns, many architecture variants are generated considering all the possible combinations of rule application from a graph grammar. Simulation and optimization allow to assess the generated architectures in order to select the best one according to the architect's directives. In addition, the architect uses timed temporal logic to define static and dynamic contracts in order to specify acceptable architectures and evolution of the system of systems [7]. While this method addresses the issues of the target architecture and of the reconfiguration specification, to the best of our knowledge, the DANSE

[2]Infrastructure Nationale Partageable des Transmissions http://crd.ensosp.fr/index.php?lvl=notice_display&id=9395

Figure 4: French Emergency SoS



project does not prescribe how the reconfiguration is effectively built. In this regard, we think that our work is orthogonal to the proposals of the DANSE project.

Several previous work draw general principles to address dynamic reconfiguration. Among them, designed for user interfaces, C2 [17] is an architectural style in which the components are organized in layers separated by connectors. C2 assigns to the connectors a key role for dynamic reconfiguration. Since the connectors are responsible of the message routing policy, any reconfiguration action actually occurs within the connectors. While C2 is not designed for system of systems, its organization with respect to dynamic reconfiguration provides a potential solution to overcome the lack of control over the constituents. In a different context, Boyer *et al* [2] proposed a formally-verified algorithm in order to issue a valid sequence of reconfiguration actions, even in presence of component failures. The reconfiguration is split in two phases: in the *down* phase, components are stopped, unwired and destructed; in the *up* phase, they are created, wired and started. Whenever failures are detected in the process, the effects of failures are first propagated, *i.e.*, failed components are unwired, then the reconfiguration process resumes to either recover the initial architecture or reach the target one. A similar principle may address the case when constituents choose to disengage from a system of systems.

Other works define and analyze reconfiguration with architectural styles [18, 5]. They focus on how the architectural style affects reconfiguration. In none of these works, the step towards capturing general principles for solving the reconfiguration problem has been done. Though, their underlying principles are of interest in the SoS area.

To build the standard form, our inspiration comes from the work on patterns, including in other areas. Gomaa *et al* [10, 9] collect several patterns targeted at reaching quiescence noticeably in client/server, master/slave and service-oriented architectures. On the one hand, quiescence is no more no less one possible solution to the specific problem of lowering the coupling for the time of reconfiguration. We think that reconfiguration patterns should have a wider scope. On the other hand, the standard form of their patterns contains only the state-charts that relate the operational states to the quiescent ones. They do not give an explicit statement of the addressed problem, nor do they provide explicit assumptions or properties of the patterns.

Ramirez *et al* [14] adopt a more complete form derived from the one of Gamma *et al* [8] with an additional LTL-based constraint entry. Their catalog mixes design patterns to build monitoring and decision-making subsystems, as well as reconfiguration patterns. However, having a single common form for everything leads to omit the specificities of each area. For instance, their *component insertion* and *component removal* patterns make undocumented assumptions on the reconfiguration grammar that can hardly be guessed, and which appears incompatible with the system of systems approach. Our work may be less general, but, being specifically targeted at reconfiguration, it does not suffer such drawbacks.

In summary, in the current state of the art, the problem of designing and building reconfigurations that accommodate the unique characteristics of systems of systems has still to be addressed. Some work proposed general principles that could be helpful in the context of systems of systems. However, to be really reusable, their presentation must be homogenized, like it is done for patterns in general. While existing pattern catalogs in other areas gives insights, their usual presentation does not fit the specificities of reconfiguration patterns.

5. CONCLUSION

In summary, we think that reconfiguration requires well documented design: we suggest a pattern approach. An important characteristic to be documented is the heterogeneity of the reconfiguration behavior of system. Moreover, reconfiguration patterns for SoS should document how to deal with the lack of coercion on systems. This characteristics comes from operational and managerial independence. The emergence characteristic regarding reconfiguration is not specific. It involves to preserve particular configuration during reconfiguration. This aspect is tackled by architectural invariants and behavioral properties.

We consider that reconfiguration patterns have different granularity. Coarse-grained patterns describe overall and global reconfiguration strategies. Fine-grained patterns describe solutions to specific reconfiguration issues. As consequence coarse-grained patterns identify technical issues (variability point) and leave architecture to solve them with fine-grained pattern for instance.

In this paper we proposed a new approach to address reconfiguration, taking into account the specific considerations of SoS: we extend the pattern concept to reconfiguration. Our underpinning idea consists in documenting reconfiguration and providing principled solutions to well-identified reconfiguration problems. These patterns are adaptable building blocks that can be reused in different systems which share a similar context and similar objectives, hence lowering design time of SoS reconfiguration. With explicit architectural invariants and behavioral properties, the patterns make it clear how the system behaves even during reconfiguration. The architectural invariants assist the architect in providing specifications to the reconfiguration.

The patterns assist the design step. They describe problems involved by reconfiguration in a particular architectural context and a solution.

This documentation will provide with solid ground in the perspective of the analysis and verification of reconfigurations.

From the verification perspective, patterns provide models. As consequence engineering effort to check a reconfiguration can be reused in other similar contexts. From a validation perspective, the solution provides the operations that led to maintain the invariant set in the problem.

In our future work, we intend to build a pattern system, in the sense of [4]. Namely, it will be a collection of patterns, which defines a language, along with guidelines on how this language should be used in practice to design and build reconfigurations. We foresee that this pattern system will address several classes of reconfiguration problems, ranging from effective changes to their management and coordination. We ultimately aim at addressing all of them consistently in a single pattern system.

6. REFERENCES

[1] Arshad, N., Heimbigner, D., Wolf, A.L.: Deployment and dynamic reconfiguration planning for distributed software systems. Software Quality Journal 15(3), 265–281 (2007)

[2] Boyer, F., Gruber, O., Pous, D.: Robust reconfigurations of component assemblies. In: Proceedings of the 2013 International Conference on Software Engineering. pp. 13–22. IEEE Press (2013)

[3] Buisson, J., Dagnat, F., Leroux, E., Martinez, S.: Safe reconfiguration of coqcots and pycots components. Journal of Systems and Software pp. – (2015), http://www.sciencedirect.com/science/article/pii/S0164121215002630

[4] Bushchmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M.: Pattern-oriented software architecture: a system of patterns, vol. 1. Wiley (1996)

[5] Dorn, C., Taylor, R.N.: Analyzing runtime adaptability of collaboration patterns. Concurrency and Computation: Practice and Experience 27(11), 2725–2750 (2015)

[6] Eckardt, T., Heinzemann, C., Henkler, S., Hirsch, M., Priesterjahn, C., Schäfer, W.: Modeling and verifying dynamic communication structures based on graph transformations. Comput. Sci. 28(1), 3–22 (Feb 2013), http://dx.doi.org/10.1007/s00450-011-0184-y

[7] Etzien, C., Gezgin, T., Froschle, S., Henkler, S., Rettberg, A.: Contracts for evolving systems. In: Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC), 2013 IEEE 16th International Symposium on. pp. 1–8 (June 2013)

[8] Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-oriented Software. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (1995)

[9] Gomaa, H., Hashimoto, K., Kim, M., Malek, S., Menascé, D.A.: Software adaptation patterns for service-oriented architectures. In: Proceedings of the 2010 ACM Symposium on Applied Computing. pp. 462–469. SAC '10, ACM, New York, NY, USA (2010), http://doi.acm.org/10.1145/1774088.1774185

[10] Gomaa, H., Hussein, M.: Software reconfiguration patterns for dynamic evolution of software architectures. In: Software Architecture, 2004. WICSA 2004. Proceedings. Fourth Working IEEE/IFIP Conference on. pp. 79–88. IEEE (2004)

[11] Ingram, C., Payne, R., Perry, S., Holt, J., Hansen, F.O., Couto, L.D.: Modelling patterns for systems of systems architectures. In: Systems Conference (SysCon), 2014 8th Annual IEEE. pp. 146–153. IEEE (2014)

[12] Kalawsky, R.S., Joannou, D., Tian, Y., Fayoumi, A.: Using architecture patterns to architect and analyze systems of systems. Procedia Computer Science 16, 283–292 (2013)

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

[14] Ramirez, A.J., Cheng, B.H.C.: Design patterns for developing dynamically adaptive systems. In: Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems. pp. 49–58. SEAMS '10, ACM, New York, NY, USA (2010), http://doi.acm.org/10.1145/1808984.1808990

[15] Schmidt, D.C., Stal, M., Rohnert, H., Buschmann, F.: Pattern-Oriented Software Architecture, Patterns for Concurrent and Networked Objects, vol. 2. John Wiley & Sons (2013)

[16] Simonot, M., Aponte, V.: A declarative formal approach to dynamic reconfiguration. In: Proceedings of the 1st International Workshop on Open Component Ecosystems. pp. 1–10. IWOCE '09, ACM, New York, NY, USA (2009), http://doi.acm.org/10.1145/1595800.1595802

[17] Taylor, R.N., Medvidovic, N., Anderson, K.M., Whitehead, Jr., E.J., Robbins, J.E.: A component- and message-based architectural style for gui software. In: Proceedings of the 17th International Conference on Software Engineering. pp. 295–304. ICSE '95, ACM, New York, NY, USA (1995), http://doi.acm.org/10.1145/225014.225042

[18] Taylor, R.N., Medvidovic, N., Oreizy, P.: Architectural styles for runtime software adaptation. In: Software Architecture, 2009 & European Conference on Software Architecture. WICSA/ECSA 2009. Joint Working IEEE/IFIP Conference on. pp. 171–180. IEEE (2009)

[19] Winokur, M., Goldberg, R., Dov, N.B., Mangeruca, L., Passerone, R., Senni, V., Etzien, C., Gezgin, T., Peikenkamp, T., Jung, M., Alexandre, A., Bullinga, R., Imad, S., Honour, E., Paul, S., Klaas, S., Boyer, B., Kemper, S.: Danse methodology v03. deliverable D_4.4, DANSE (Feb 2015), http://www.danse-ip.eu/home/pdf/danse_d4.4_danse\%20methodology.pdf, consulted on 02/22/2016