HAL
open science

Une Approche Orientée Pattern pour la Reconfiguration de Système de Systèmes

Franck Petitdemange, Jérémy Buisson, Isabelle Borne

To cite this version:

Franck Petitdemange, Jérémy Buisson, Isabelle Borne. Une Approche Orientée Pattern pour la Reconfiguration de Système de Systèmes. CIEL 2015, Jun 2015, Bordeaux, France. hal-01421540

HAL Id: hal-01421540

https://hal.science/hal-01421540v1

Submitted on 22 Dec 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Une Approche Orientée Pattern pour la Reconfiguration de Système de Systèmes

Petitdemange Franck[1], Buisson Jérémy[2], and Borne Isabelle[1]

[1] IRISA Université de Bretagne, Vannes, France
franck.petitdemange@irisa.fr, isabelle.borne@irisa.fr
[2] IRISA Académie de St-Cyr, Vannes, France
jeremy.buisson@irisa.fr

Résumé

Les Systèmes de Systèmes (SoS) sont une classe particulière de systèmes composés de constituants géographiquement distribuées, hétérogènes et sujets à évolution. Généralement les SoS sont construits dynamiquement et recrutent leurs constituants à l'exécution. Afin qu'un SoS puisse continuer à remplir sa mission globale, il est nécessaire qu'il se reconfigure au fur et à mesure des évolutions de son environnement ou de l'engagement ou désengagement de ses constituants. Nous proposons de traiter la reconfiguration d'un SoS au niveau de son architecture, et nous nous plaçons dans le cas où l'architecture est décrite à l'aide de patterns d'architecture. Notre objectif est de définir et d'outiller des patterns de reconfiguration qui décrivent les opérations permettant de maintenir le SoS dans un état cohérent par rapport aux patterns d'architecture qui le décrivent. Nous présentons la première étape de notre travail, consacrée à la modélisation d'exemples de reconfigurations, issus de l'étude d'un cas réel, en utilisant les concepts d'un langage de description des architectures de SoS.

Abstract

Systems-of-systems (SoS) are a particular class of systems that are composed of constituents geographically distributed, heterogeneous and evolutionary. Usually, SoS are dynamically built and recruit their constituents at runtime. In order to achieve its global goal, it needs to reconfigure according to the evolutions of its environment or the commitment or disengagement of its constituents. We propose to deal with SoS reconfiguration at its architectural level, and in the case of an architectural pattern based architecture. Our purpose is to define reconfiguration patterns which describe the operations allowing to maintain the SoS in a coherent state, with regard to the SoS architectural patterns. We present the first step of our work devoted to model reconfiguration examples extracted from a real case, and by using concepts of an SoS architecture description language.

1 Introduction

La complexité croissante des systèmes à logiciel prépondérant a fait émerger ces dernières années le concept de système de systèmes [7]. Il s'agit d'une classe particulière de systèmes composés de constituants géographiquement distribués, hétérogènes qui collaborent pour remplir une mission, et qui sont sujets à évolution. La plupart du temps les systèmes constituants sont des systèmes qui ont été conçus pour remplir des missions particulières indépendamment d'un SoS dans lequel ils pourront être intégrés plus tard. Par ailleurs, le manque de pouvoir coercitif sur l'environnement d'un SoS entraîne le fait que des détails architecturaux d'un SoS peuvent être reportés à l'exécution. Dans ce contexte notre projet a pour objectif de réaliser des opérations de reconfiguration dynamique décrites par des patterns et outillées à l'aide d'un langage de description d'architecture de SoS (SoSADL) en cours de développement dans notre équipe.

La section 2 est consacrée à la description d'un exemple de SoS extrait d'une étude de cas réelle qui nous servira plus tard à valider notre travail. La section 3 montre les besoins de

reconfiguration et en quoi des patterns de reconfiguration vont nous aider en s'appuyant sur l'exemple décrit dans la section précédente. Nous terminons par un survol de travaux connexes et les travaux futurs que nous envisageons.

2 Exemple illustratif

Pour illustrer nos propos nous avons identifié un système de surveillance des inondations mis en place dans la région de São Carlos au Brésil [3] comme un Système de Systèmes (SoS). Ce SoS assiste la surveillance de rivières et la génération de messages d'alerte pour notifier les autorités et les citoyens sur un risque d'inondation imminente. Le système de surveillance des inondations est composé d'un ensemble de capteurs structuré en grappes. Chaque capteur peut effectuer des mesures environnementales (niveau rivière, vitesse) et combiner ces données pour surveiller et/ou anticiper des phénomènes naturels. Les capteurs sont répartis sur différents sites identifiés à risque. Chaque capteur possède des capacités de calcul, mémoire, transmission et énergie leur procurant une certaine autonomie. Chaque site surveillé possède un collecteur qui a des capacités de communication hors-site, une ressource ¡¡ illimitée ¿¿ en énergie et des capacités de calcul. Chaque capteur est atteignable par un collecteur.

Ce système répond aux propriétés identifiées dans la littérature pour les SoS : (i) géographiquement distribué, les capteurs sont placés le long des rivières (ii) développement évolutionnaire, des capteurs d'un nouveau type de peuvent s'intégrer au cours du temps (iii) comportement émergent, l'interaction des capteurs permet de surveiller et prévoir des phénomènes environnementaux que chaque capteur ne peut évaluer indépendamment (iv) indépendance opérationnelle, chaque capteur effectue des mesures environnementales de façon autonome (v) indépendance managériale : chaque capteur appartient à une organisation qui contrôle et développe le cycle de vie du capteur.

Les capteurs possèdent leur propre indépendance managériale et opérationnelle mais sont conçus dans le but de remplir les missions définies par le SoS. Le SoS est de type dirigé.

2.1 Description formelle

La description d'un SoS s'articule autour de trois niveaux d'abstraction : mission, architecture abstraite, architecture contrète. Comme tout système, un SoS satisfait un but global qui est spécifié en terme de missions qui définissent comment le SoS doit réaliser le but indépendamment des détails d'implémentation. À partir de l'interprétation des missions, l'architecte définit une architecture abstraite (ou *coalition* en SoSADL) qui identifie les rôles permettant de satisfaire la mission globale. En SoSADL le SoS coordonne ses constituants par des éléments d'interaction nommés *médiateurs*. Basé sur cette architecture abstraite, le SoS génère une architecture concrète respectant les contraintes définies par l'architecture abstraite selon les systèmes (détectés à l'exécution) pouvant jouer chaque rôle identifié dans l'architecture abstraite.

2.1.1 Mission

La mission définie dans le SoS est la prévision d'un risque d'inondation. Cette mission est raffinée en plusieurs sous-missions pour prédire un risque : évaluation de la pente de l'eau, de la vélocité, et de la largeur du fleuve. Chacune de ces sous-missions définit une mission de routage vers le système constituant (CS) de calcul de l'indice de risque. Une contrainte sur les propriétés de routage peut être spécifiée (e.g. préserver le niveau de batterie).

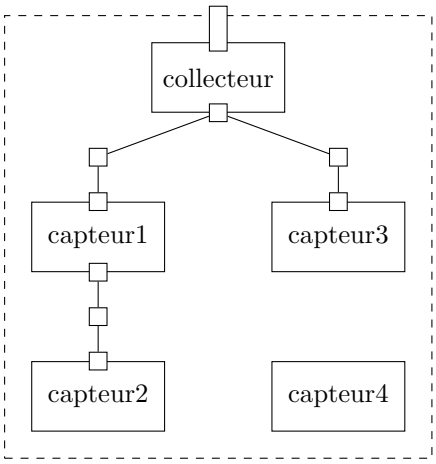

FIGURE 1 : Architecture concrète avec contrainte plus court chemin

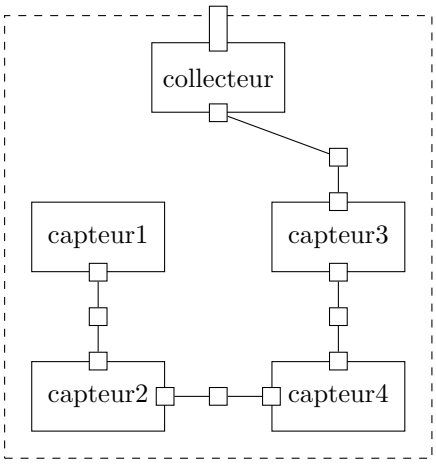

FIGURE 2 : Architecture concrète avec contrainte économie d'énergie

2.1.2 Architecture abstraite

L'architecture abstraite de notre exemple de SoS suit un pattern architectural logiciel : Event-Based Publish-Subscribe Le modèle conceptuel définit trois types d'artefacts : *Bus de communication*, *Publisher* et *Subscriber*. Ce pattern est utilisé dans les systèmes disséminant des événements (informations) de façon asynchrone et décentralisée. La coalition qui modélise l'architecture abstraite identifie que le SoS doit posséder des CSs pour mesurer la vélocité de l'eau, la pente et la largeur de la rivière et pour finir un CS pour calculer l'indice de risque. Cela définit les rôles des CSs dans le SoS. Les médiateurs définis dans la coalition sont chargés du rôle de bus de communication. Ils encapsulent les détails de communications comme la correspondance de nom vers les localisations physiques. Ils ont la responsabilité de recevoir et/ou relayer des événements. Les médiateurs organisent la topologies du bus de communication suivant les contraintes de préservation d'énérgie ou de latence.

2.1.3 Architecture Concrète

L'architecture concrète est une instance particulière de l'architecture abstraite qui satisfait toutes les contraintes. Elle sélectionne et compose à l'aide de médiateurs les systèmes qui contribuent à la réalisation de la mission. La figure 1 décrit une architecture concrète pour laquelle le SoS choisit des capteurs ayant la capacité d'accomplir les fonctionnalités attendues par les missions définies et instancie une topologie qui garantit que le chemin vers le CS de calcul est le plus court possible. La figure 2 présente une autre topologie qui minimise la consommation d'énergie.

3 Pattern de reconfiguration

L'exemple de la section 2 pointe plusieurs besoins de reconfiguration. Le premier est la résilience des patterns architecturaux définis et des missions associées. Par exemple, un collecteur doit toujours recevoir les événements environnementaux spécifiés pour calculer régulièrement un indice

de risque. Un second besoin est la modification d'une mission, par exemple, l'ajout de contraintes sur une mission, pour l'optimisation des propriétés non-fonctionnelles (e.g. préservation de l'énergie, diminution du temps de latence). Un troisième besoin peut être l'application d'un nouveau pattern sur l'architecture abstraite résultant de changements dans l'environnement. Par exemple, l'évolution des capteurs vers un pattern pour la distribution de calcul. En effet, le SoS peut recruter de façon opportuniste un système vidéo. L'application d'algorithme de traitement d'image peut amener l'évolution du SoS vers ce type de pattern.

Les objectifs de la reconfiguration sont la satisfaction des objectifs du SoS tout en minimisant les temps d'arrêt, localisant les impacts d'une reconfiguration et maintenant la cohérence du système. Dans les SoS, l'évolution constante est une caractéristique clé. Elle concerne les trois niveaux d'abstraction définis dans le section 2.1, or c'est un aspect qui n'est pas complétement abordé dans les patterns architecturaux classiques. Par exemple [1, 5] proposent d'appliquer des reconfigurations respectant un modèle de composant, mais ne proposent pas de solution pour gérer l'évolution de l'architecture concrète pendant l'application des reconfigurations.

D'après les besoins de reconfiguration nous pouvons identifier deux cas de reconfiguration : (i) le premier concerne la résilience des patterns adoptés par le SoS. Par exemple, si un capteur quitte le SoS, celui-ci doit être capable d'appliquer des opérations cohérentes pour la résolution des transactions échouées, réorganiser les CS et médiateurs pour satisfaire les contraintes structurelles et continuer d'accomplir la mission. D'après la figure 2 si le capteur 3 se déconnecte, le SoS doit s'assurer que les événements des capteurs 1 et 2 ont bien été transmis au collecteur ; remplacer le capteur 3 déconnecté par le 4 si celui-ci peut remplir les mêmes fonctions ; puis réorganiser la topologie pour satisfaire la contrainte d'économie d'énergie. (ii) le second concerne l'application d'un nouveau pattern (e.g. évolution ou optimisation). D'après l'exemple de la figure 2, si on considère une évolution vers une grille de calcul, il faut déterminer un état de *quiescence* qui assure que tous les événements ont bien été transmis au collecteur ; définir l'attribution de fonctions au CS ; réinstancier de nouveaux médiateurs capables de gérer un nouveau mode de communication.

Plusieurs facteurs interviennent pour la résolution des contraintes de la coalition à instancier. Les reconfigurations doivent prendre en compte l'état de l'architecture concrète, le modèle de l'architecture abstraite et le modèle de la mission suivi par le SoS. Le lien entre les trois niveaux d'abstraction est implicite. Les patterns d'architecture et de conception sont adaptés au contexte de l'architecture concrète du SoS et à la mission définie. Dans notre approche nous souhaitons développer le concept de pattern de reconfiguration. L'idée est de rendre explicite la relation entre les trois niveaux de cohérence en déterminant les stratégies de reconfiguration à un contexte donné.

Plusieurs avantages sont attendus des patterns de reconfiguration. Ils devraient permettre de raisonner sur les choix de reconfigurations dans le SoS. Par exemple, à partir de contraintes définies par les différentes couches d'abastraction, ils permettent de comparer les différentes possibilités de reconfiguration et choisir la plus adaptée. L'aspect documentation des patterns de reconfiguration assure deux propriétés importantes : la traçabilité des patterns de reconfiguration appliqués pour faciliter l'analyse d'impact dans le SoS et guider, de façon incrémentale, l'amélioration de l'architecture abstraite du SoS ; la capitalisation de bonnes pratiques de reconfiguration.

4 Travaux connexes

Deux projets européens récents ont centré leur travail sur la conception des SoS : COMPASS et DANSE.

Du point de vue de la conception le projet COMPASS propose une collection de patterns pour guider la construction de l'architecture des SoS selon leur type (e.g. dirigé, collaboratif, etc.) [8]. Ce projet aborde également la maintenance des propriétés emergentes du SoS en proposant une approche par contrat [6]. Les contrats définissent la spécification des comportements auxquels les CSs doivent se conformer. Les auteurs proposent plusieurs techniques de test pour vérifier le respect des contrats par les CSs.

Le projet DANSE [4] propose un processus de développement incrémental des architectures de SoS reposant sur des patterns d'architectures. Les patterns sont utilisés pour générer des architectures optimisées qui satisfont les buts du SoS.

Ces deux projets n'abordent pas les problèmes liés à la reconfiguration dynamique dans les architectures de SoS. À notre connaissance peu de travaux abordent cet aspect et en particulier du point de vue SoS. Du point de vue de la reconfiguration dynamique, des propositions [1, 5] adressent uniquement la préservation d'invariants au niveau des opérations primtives et dans des modèles de composants.

5 Travaux futurs

Notre objectif est d'assister et d'outiller la reconfiguration d'architectures de SoS à l'aide de patterns. Pour des SoS implémentant des patterns d'architecture, notre approche suggère un ensemble de patterns de reconfiguration à appliquer pour maintenir la cohérence de l'architecture du SoS dans le cas d'une évolution et ce sans interrompre l'exécution du système. Dans ce papier nous avons montré sur un exemple deux cas de reconfiguration. L'étape suivante consistera à identifier dans d'autres exemples de SoS d'autres cas de reconfiguration récurrents et à capturer leur solution générale sous forme de patterns et en termes de SoSADL. L'ensemble des patterns extraits pourront constituer un langage de patterns pour la reconfiguration des SoS. Les langages de patterns facilitent la construction d'un système en décrivant les relations possibles entre différents patterns pour une famille de systèmes donnée.

Une perspective à plus long terme sera l'étude et l'assistance de la composition des patterns définis [9] avec la conception d'outils vérifiant dynamiquement les possibilités de composition et guidant le génération et la coordination des opérations de reconfiguration.

Références

[1] Jérémy Buisson, Everton Calvacante, Fabien Dagnat, Elena Leroux, and Sébastien Martinez. Coqcots & pycots : Non-stopping components for safe dynamic reconfiguration. In *Proceedings of the 17th International ACM Sigsoft Symposium on Component-based Software Engineering*, CBSE '14, pages 85–90, New York, NY, USA, 2014. ACM.

[2] Frank Buschmann, Kevlin Henney, and C. Schmidt Douglas. *Pattern-oriented software architecture, volume 4 : A Pattern Language for Distributed Computing*, volume 4. John Wiley and Sons, 2007.

[3] Danny Hughes, Jó Ueyama, Eduardo Mendiondo, Nelson Matthys, Wouter Horré, Sam Michiels, Christophe Huygens, Wouter Joosen, Ka Lok Man, and Sheng-Uei Guan. A middleware platform to support river monitoring using wireless sensor networks. *Journal of the Brazilian Computer Society*, 17(2) :85–102, June 2011.

[4] R.S. Kalawsky, D. Joannou, A. Bhatt, K. Ramalingam, I. Sanduka, M. Masin, E. Shindin, and U.Shani. Report on danse architectural approaches. Technical report, DANSE project, 2014.

[5] Marc Léger, Thomas Ledoux, and Thierry Coupaye. Reliable dynamic reconfigurations in a reflective component model. In *Proceedings of the 13th International Conference on Component-Based Software Engineering*, CBSE'10, pages 74–92, Berlin, Heidelberg, 2010. Springer-Verlag.

[6] Wen ling Huang, Jan Peleska, Ken Pierce, and Uwe Schulze. Contract support for evolving sos. Technical report, COMPASS Project, 2014.

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

[8] Simon Perry, Jon Holt, Richard Payne, Jeremy Bryans, Claire Ingram, Alvaro Miyazawa, Stefan Hallerstede, Luis D Couto, Anders Kaels Malmos, Juliano Iyoda, Marcio Cornelio, and Jan Peleska. Final report on sos architectural models. Technical report, COMPASS Project, 2014.

[9] Minh Tu Ton That, Salah Sadou, Flávio Oquendo, and Isabelle Borne. Composition-centered architectural pattern description language. In *Software Architecture - 7th European Conference, ECSA 2013, Montpellier, France, July 1-5, 2013. Proceedings*, pages 1–16, 2013.