Managing Conflicting Requirements in Systems of Systems

Thiago Affonso de Melo Novaes Viana
thiago.viana@open.ac.uk
School of Computing and Communications/ Faculty of Science, Technology, Engineering and Mathematics

Supervisors name/s: Prof. Andrea Zisman and Dr. Arosha Bandara
Starting date: 01/02/2016 (Full-Time)

I. Introduction

Software systems have evolved from being stand-alone systems to being composed into Systems of Systems. A System of Systems (SoS) is defined as an arrangement of independently created, discovered, and selected systems, which are integrated into a single system in order to deliver unique capabilities [1]. In this context, each participating system can operate and support different goals in its own environment (viz. *local goals*), as well as support new goals of the SoS as a whole (viz. *global goals*), that could not be achieved separately by the participating systems. An SoS presents many features including operational and managerial independence, geographic distribution of participating systems, and emergent behaviors [2]. Examples of SoSs are found in several different domains including, but not limited to, transport network systems, household energy management systems, personal nutrition management systems, smart homes, smart cities and intelligent healthcare systems.

However, as the SoS is formed by the integration of independent complex systems, this will increase the complexity of the SoS to at least one more order than its component systems [3]. This means that problems in the SoS environment are harder to handle than in the component system environment. Therefore, this will bring a number of software engineering challenges regarding their specification, design, construction, and operation. Among these, one important challenge is concerned with managing emerging conflicting behavior that leads to inconsistency. In an SoS, the various participating systems are often from different domains; are developed by different teams of people under different circumstances and time; have distinct functionalities; and are used by different stakeholders. All of these factors contribute to the existence of inconsistent and conflicting requirements.

This research presents an overview of a framework called MaCoRe_SoS (**Ma**naging **Co**nflicting **Re**quirements in **S**ystems **o**f **S**ystems) that supports conflict management in SoSs. The management of conflicting requirements in the proposed framework involves three main steps, namely (a) *conflict identification*, (b) *conflict diagnosis*, and (c) *resolution* based on the use of a utility function. In order to support the main steps, the framework uses a Monitor-Analyze-Plan-Execute-Knowledge (MAPE-K) architectural pattern[4].

II. Motivating Example

In order to illustrate and evaluate the proposed framework, this research uses *FeedMe FeedMe* [5], an exemplar of an SoS scenario composed of different systems to address food security problems at different levels of granularity, namely individuals, groups, cities and nations. At the individual level (viz. *AnalyseMe*), *FeedMe FeedMe* presents smart devices to monitor, analyse and provide suggestions about the nutritional and health status of a person. At the group level (viz. *HomeHub*), *FeedMe FeedMe* uses smart home appliances that interoperate to create a more precise family meal plan, based on the family resources and budget. At the city level (viz. *SmartCity*), local markets collect data from multiples families to manage their stock and to reduce food wastage. At the national level (viz. *SmartNation*), food producers and manufacturers collect data from different markets to forecast food needs and provide alternatives in case of food crisis. Figure 1 shows an overview of the *FeedMe FeedMe* SoS with its various participating systems and devices.

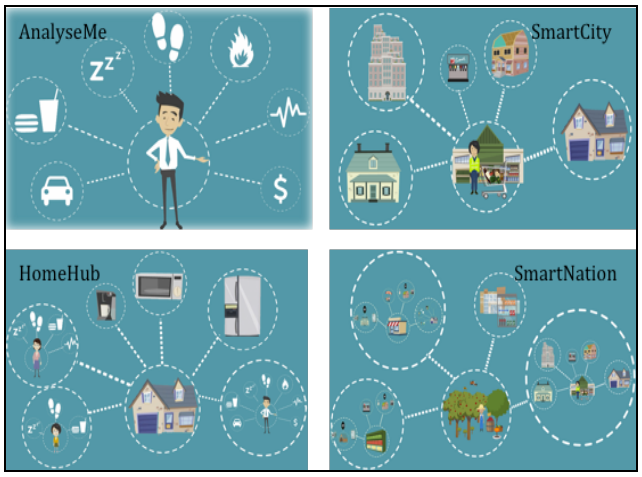

Fig. 1. Overview of the FeedMe FeedMe SoS

III. The MaCoRe_SoS Framework

The main goal of the *MaCoRe_SoS* framework is to manage conflicting requirements in SoSs. In the framework, the requirements represent both local goals of the participating components and global goals of the SoS environment as a whole. We distinguish these as *local* and *global requirements*.

Figure 2 shows an overview of the MaCoRe_SoS framework. As shown in the figure, the framework uses a *conflict manager component* to supports requirements conflict management based on three main steps, as described in the

work from Spanoudakis and Zisman [6], namely, (i) conflict identification (ii) conflict diagnosis, and (iii) conflict resolution. It supports SoS environments composed of other stand-alone component sub-systems (CS), services, or even other systems of systems. For simplicity, we will refer to a participating component sub-system, service, or SoS, as an *entity*.

Each participating entity registers in an SoS and provides its respective requirement specifications and an ontology that is used by the framework to represent concepts of the domain associated with the entity. The ontologies are integrated into a shared ontology in order to assist with the identification of elements that are shared by the various participating entities during the overlap detection and conflict detection activities.

Moreover, as SoSs operate in dynamic environments where the satisfaction of requirements depends on runtime states that are uncertain at design time. The *MaCoRe_SoS* framework assumes requirements specified using fuzzy branching temporal logic (FBTL) [7] under a structured representation of the RELAX language [8], which has specific support for uncertain in the relationship between requirements and the environment.

The conflict identification, diagnosis, and resolution steps in the framework are executed based on the Monitor-Analyze-Plan-Execute-Knowledge (MAPE-K) architectural pattern [4]. The overlap detection is executed using ontologies and identifies requirements that share common elements, such as resources. The identification of conflicts is assisted by a monitor component that detects violations in the requirements at runtime. The diagnosis of the conflicts is performed by an analyzer component using requirements interaction features [9]. The resolution of conflicts is based on the use of a configurable utility function and supports eight resolution methods: relaxation, refinement, abandonment, compromise, restructuring, reinforcement, re-planning, and postponement [9].

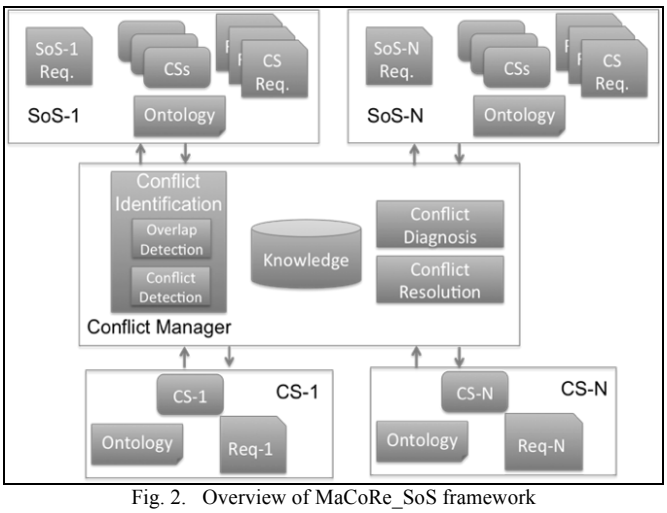

Fig. 2. Overview of MaCoRe_SoS framework

The framework also includes a database that stores necessary knowledge used during conflict management (e.g., historical data about resolution strategies used in previous conflict resolutions and information about requirements violations).

IV. Conclusion

The growth in the complexity and heterogeneity of modern software systems has led to systems that compose themselves into bigger systems to achieve more sophisticated functionalities. These systems are often System of Systems (SoS) where the management of emerging conflicting behaviors, expressed as requirements is a challenge. As a new and emergent application, the management of inconsistencies is an important task inside the SoS environment.

Therefore, as different systems are composed together, emergent and undesirable behaviors arise, leading to conflicting requirement. To address this problem, we present the *MaCoRe_SoS* framework, with three steps: (a) conflict identification, (b) conflict diagnosis, and (c) resolution.

We have built a prototype version of the framework and demonstrated its efficacy for scenarios based on FeedMe FeedMe, an example SoS ecosystem designed to support food security at different levels of granularity (individuals, families, cities, and nations). The initial results demonstrate that it is possible to identify and manage conflicts and that the application of a resolution method is able to support the SoS to satisfying local and global requirements.

References

[1] Office of the Deputy Under Secretary of Defense for Acquisition and Technology, Systems and Software Engineering, *Systems Engineering Guide for Systems of Systems*, vol. 1. Washington, DC: ODUSD(A&T)SSE, 2008.

[2] M. W. Maier, ‘Architecting principles for systems-of-systems’, in *INCOSE International Symposium*, 1996, vol. 6, pp. 565–573.

[3] A. P. Sage and C. D. Cuppan, ‘On the systems engineering and management of systems of systems and federations of systems’, *Inf. Knowl. Syst. Manag.*, vol. 2, no. 4, pp. 325–345, 2001.

[4] J. O. Kephart and D. M. Chess, ‘The vision of autonomic computing’, *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[5] A. Bennaceur, C. Mccormick, J. Garc__a Gal_an, C. Perera, A. Smith, et and al, ‘Feed me, Feed me: An Exemplar for Engineering Adaptive Software’, presented at the 11th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, Austin, United States, 2016.

[6] G. Spanoudakis and A. Zisman, ‘Inconsistency management in software engineering: Survey and open research issues’, *Handb. Softw. Eng. Knowl. Eng.*, vol. 1, pp. 329–380, 2001.

[7] S. Moon, K. H. Lee, and D. Lee, ‘Fuzzy branching temporal logic’, *IEEE Trans. Syst. Man Cybern. Part B Cybern.*, vol. 34, no. 2, pp. 1045–1055, 2004.

[8] J. Whittle, P. Sawyer, N. Bencomo, B. H. Cheng, and J.-M. Bruel, ‘RELAX: a language to address uncertainty in self-adaptive systems requirement’, *Requir. Eng.*, vol. 15, no. 2, pp. 177–196, 2010.

[9] W. N. Robinson, S. D. Pawlowski, and V. Volkov, ‘Requirements interaction management’, *ACM Comput. Surv. CSUR*, vol. 35, no. 2, pp. 132–190, 2003.