Identifying Conflicting Requirements in Systems of Systems

Thiago Viana
Open University
School of Computing and Communications
Milton Keynes, UK
thiago.viana@open.ac.uk

Andrea Zisman
Open University
School of Computing and Communications
Milton Keynes, UK
andrea.zisman@open.ac.uk

Arosha K. Bandara
Open University
School of Computing and Communications
Milton Keynes, UK
arosha.bandara@open.ac.uk

***Abstract*— A System of Systems (SoS) is an arrangement of useful and independent sub-systems, which are integrated into a larger system. Examples are found in transport systems, nutritional systems, smart homes and smart cities. The composition of component sub-systems into an SoS enables support for complex functionalities that cannot be provided by individual sub-systems on their own. However, to realize the benefits of these functionalities it is necessary to address several software engineering challenges including, but not limited to, the specification, design, construction, deployment, and management of an SoS. The various component sub-systems in an SoS environment are often concerned with distinct domains; are developed by different stakeholders under different circumstances and time; provide distinct functionalities; and are used by different stakeholders, which allow for the existence of conflicting requirements. In this paper, we present a framework to support management of emerging conflicting requirements in an SoS. In particular, we describe an approach to support identification of conflicts between *resource-based requirements* (i.e. requirements concerned with the consumption of different resources). In order to illustrate and evaluate the work, we use an example of a pilot study of an IoT SoS ecosystem designed to support food security at different levels of granularity, namely individuals, groups, cities, and nations.**

***Index Terms*—Conflicting requirements, Systems of Systems, Conflict Identification**

I. Introduction

A System of Systems (SoS) is a collection of sub-systems providing: operational and managerial independence of the sub-systems, geographical distribution of the sub-systems, emergent behavior, and evolutionary development processes [1]. Examples of SoSs are found in several areas such as transportation, nutrition management, smart homes, smart cities, etc. In such environments, the various participating sub-systems in an SoS 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. An important challenge in SoSs is concerned with the management of inconsistent emerging requirements. More specifically, in an SoS, the various participating sub-systems may present conflicting requirements among themselves, as well as emerging conflicting requirements between the whole SoS and the participating sub-systems. For example, consider a requirement of a smart home SoS concerned with the reduction of electricity consumption in the home, and a requirement of the meal-planner sub-system in the smart home SoS to keep information about daily food intake of all the family members. In this case, the meal-planner requirement consumes a large amount of electricity to be able to keep the necessary information and, therefore, it conflicts with the electricity consumption reduction in the home.

In this paper we present a framework called *MaCoRe_SoS (**Ma**naging **Co**nflicting **Re**quirements in **S**ystems **of S**ystems)* to support conflict management in SoSs. In particular, we are interested in identifying conflicting requirements associated with the consumption of *resources* by the participating sub-systems in an SoS (i.e., *resource-based requirements*). The different types of resources depend on the domains of the participating systems and the SoS as a whole. For instance, in the case of a smart home nutrition management SoS, relevant resources would include the calorific content of meals or the daily calories consumed by an individual; the quantity and cost of ingredients in meals; energy consumption to prepare meals; and individuals' insulin, cholesterol or blood pressure levels. In the case of a smart city SoS, examples of resources would include the energy consumption of households, availability of different types of transport, pollution levels, and traffic levels.

In order to address uncertainties in the SoS environment, the framework assumes requirements specified in an extension of the RELAX language [2]. The framework is composed of three steps, namely (a) *conflict identification*, (b) *conflict diagnosis*, and (c) *conflict resolution*. The conflict identification, diagnosis, and resolution steps are executed based on a Monitor-Analyze-Plan-Execute-Knowledge (MAPE-K) architectural pattern [3]. The conflict identification step is executed by two activities, namely *overlap detection* and *conflict detection*. The overlap detection is executed at design time based on the use of ontologies in order to identify requirements that share the same resources. The detection of conflicts is executed by an event monitor, which detects violations of resources at run-time. The diagnosis of the conflicts is performed by an analyzer component based on requirements interaction features. The resolution of conflicts is based on the use of a utility function and supports eight

resolution methods [4]. In this paper, we concentrate on the conflict identification step. An example of a IoT-based food security SoS called *FeedMe FeedMe* [5] is used to illustrate and evaluate the approach using a pilot study based on a simulated scenario.

The rest of this paper is structured as follows. In Section II we describe the *FeedMe FeedMe* IoT-based SoS example. In Section III we present a brief overview of the *MaCoRe_SoS* framework. In Section IV we describe the identification of conflicting requirements in the *MaCoRe_SoS* framework. In Section V we present the results of an initial evaluation using a pilot study. In Section VI we discuss related work. Finally, in Section VII we present some conclusions and future work.

II. Motivating Example

FeedMe FeedMe [5] is an IoT-based SoS exemplar composed of different independently created sub-systems to support food security issues at different levels of granularity, namely individuals, groups, cities, and nations. At the individual level, *FeedMe FeedMe* uses wearable devices to monitor, analyse and present suggestions about the nutritional and health status of an individual. At the group level, *FeedMe FeedMe* uses the interoperation of smart home appliances to create a more precise family meal plan, based on family resources and budget. At the city level, local markets collect data from various families in order to manage their stock and reduce food wastage. Finally, at the national level, food producers and manufacturers collect data from several markets to forecast food needs and provide alternatives in case of food crisis. Aligned with the granularity levels described above, the *FeedMe FeedMe* SoS consists of four different systems: AnalyseMe, HomeHub, SmartCity, and SmartNation. Figure 1 shows an overview of *FeedMe FeedMe* and its various participating systems and devices. We provide below a brief description of these systems.

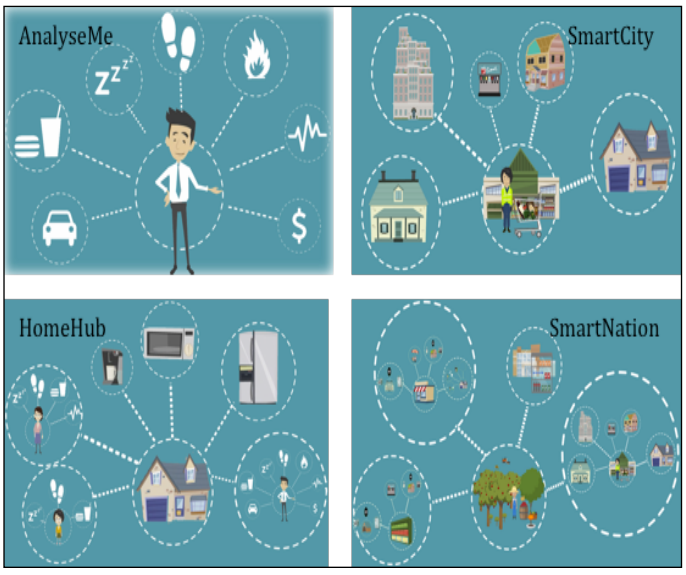

Fig. 1. Overview of FeedMe FeedMe granularity levels

AnalyseMe. This system is composed of wearable devices that collect different types of health related information from a user (e.g., heart rate, blood pressure, blood glucose, food intake, sleep and activity levels), and proposes meal and exercise plans to the user, using the collected information.

HomeHub. This system is a smart system that communicates with the home appliances in a house, aided by smart packaging applications used by refrigerator and pantry devices. This system allows family meals to be planned in advance and it is able to send a list of ingredients to supermarkets when they are required and missing from the house.

SmartCity. This system supports local supermarkets to collect information about families and communities' grocery requirements to improve the management of stocks and inventory.

SmartNation. This system aggregates requirements of individuals, family households, supermarkets, producers, manufacturers, and distributors to manage food production.

Considering that the *FeedMe FeedMe* SoS has different and independent component sub-systems, conflicting requirements may arise when requirements of the participating systems or requirements of the overall SoS cannot be satisfied due to simultaneous use of resources associated with these requirements. For example, consider a requirement of the AnalyseMe system in which meal plans that satisfy the nutritional needs of the user should be created. Consider also the fact that healthy meals are usually more expensive. Suppose another requirement of HomeHub system in which it is necessary to avoid food waste and help the user with his budget. In this case, these two requirements may conflict since the meal plan provided by the AnalyseMe could require more expensive food resources, using more budget or lead to the waste of other food resources when they do not meet the nutritional requirements. Another example is related to the fact that in order to provide a healthy lifestyle the *Feed Me Feed Me* SoS may create exercise plans to the family, however those exercise plans may request the usage of some home appliances, leading to more electricity consumption. It may conflict, for instance, with a HomeHub requirement to use as little electricity as possible.

III. An Overview of the *MaCoRe_SoS* Framework

Figure 2 shows an overview of the *MaCoRe_SoS* framework. As shown in the figure, the framework uses a *conflict manager component* to support its main steps (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 resources that are shared by the various participating entities during the overlap detection and conflict detection activities. The conflict identification, diagnosis, and resolution steps in the

framework are executed based on the Monitor-Analyze-Plan-Execute-Knowledge (MAPE-K) architectural pattern [3]. 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).

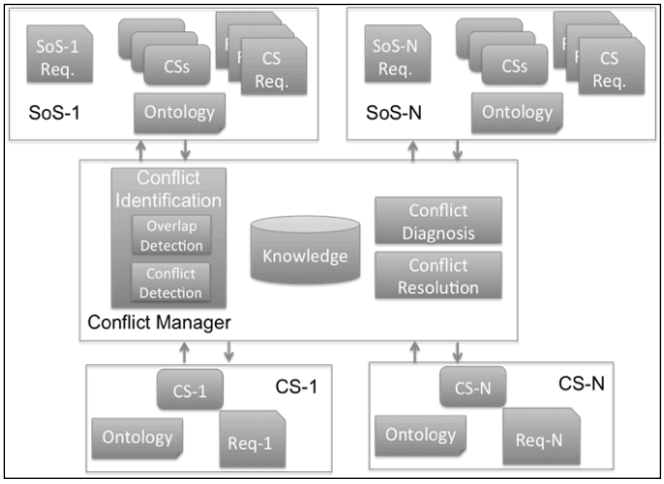

Fig. 2. Overview of *MaCoRe_SoS* framework

A monitor component assists the framework during the conflict identification step. It monitors events that are sent by the entities and how the resources are affected by these events. Also, the monitor checks the requirements satisfaction after an event, in order to detect conflicts. The diagnosis of the conflicts is performed by an analyzer component using requirements interaction features [4]. The resolution of conflicts is based on the use of a utility function and supports eight resolution methods: relaxation, refinement, abandonment, compromise, restructuring, reinforcement, re-planning, and postponement [4].

IV. Conflict Identification

The various participating entities in an SoS need to operate and support different requirements in its own environment (viz. *local requirements*), as well as support new requirements of the SoS as a whole (viz. *global requirements*), that could not be achieved separately by the participating entities. In such an environment, it is necessary to guarantee consistency between local requirements of the participating entities, as well consistency between local and global requirements. The identification of conflicting requirements in an SoS requires the monitoring of requirements at different layers and levels of granularity.

MaCoRe_SoS framework addresses the above challenges by using an extension of the RELAX language [2] to identify conflicting requirements, and by performing two activities, namely (a) overlap detection and (b) conflict detection. The first activity detects overlaps between the requirements from different entities and the resources present in the SoS environment. The second activity uses an event monitor that receives inputs from the entities and SoS actions, and checks its compliance with the requirements and the involved entities to detect violations that may lead to conflicts between requirements.

Conflicts in resource-based requirements are concerned with how resources are consumed or protected by the various participating entities and the whole SoS, as described below.

- CONSUME X CONSUME: Conflicts due to a divergent consumption of the same resource by two or more requirements.
- CONSUME X PROTECT: Conflicts due to the consumption of a resource and the prevention of the usage of the same resource by two or more requirements.
- PROTECT X PROTECT: Conflicts due to a divergent prevention of the usage of the same resource by two or more requirements.

In the following we describe the activities involved in conflict identification in more detail.

A. Requirements Representation

In self-adaptive systems, uncertainty is a common characteristic of SoSs. This is due to operational and managerial independence of each component system, and the emergent behaviors that arise in SoS environments. In order to deal with uncertainty and represent requirements, we propose to use fuzzy branching temporal logic (FBTL) [6], which supports representation of fuzziness over both statements and time. This representation provides flexibility to adapt and amend the system at runtime, and to deal with conflicting requirements.

A way to represent requirements using FBTL is with RELAX [2]. RELAX is used as a requirement specification language to represent requirements of self-adaptive systems. The vocabulary used by RELAX is based on a set of modal (e.g., SHALL, MAY … OR), temporal (e.g., EVENTUALLY, UNTIL, AS CLOSE AS POSSIBLE TO), and ordinal (e.g., AS MANY, FEW AS POSSIBLE) operators; as well as uncertainty factors (e.g., ENV, MON, REL, DEP). A full explanation of RELAX can be found in [2].

In order to illustrate our use of RELAX, consider below a local requirement of the HomeHub sub-system (HH_R1), and a global requirement (FMFM_R1) of the *FeedMe FeedMe* SoS.

HH_R1 – HomeHub SHALL control the home electricity usage to be AS CLOSE AS POSSIBLE to 100 KWh.
RESOURCE: ELECTRICITY–PROTECT
EVENT: HomeHub-SaveEnergy

FMFM_R1 – The SoS SHALL propose a family exercise plan in order to maintain each family member calories AS CLOSE AS POSSIBLE TO the ideal calories level.
RESOURCE: ELECTRICITY–CONSUME, CALORIES-PROTECT
EVENT: FMFM-FamilyExercisePlan

As shown in HH_R1 and FMFM_R1, RELAX has been extended with the RESOURCE and EVENT clauses. The RESOURCE clause represents the type of resource associated with the requirement, and how this resource should be used by that requirement. For example, a resource type can be consumed or protected. In the case of being consumed (CONSUME), the associated value of the resource is decreased. In the case of being protected (PROTECT), the consumption of the associated value of the resource should be prevented. The

RESOURCE clause can accommodate the representation of more than one type of resource associated with the requirement. The EVENT clause represents the different types of events that will trigger the associated requirement. It is possible to have the same event associated with different requirements, and a requirement triggered by different types of events. The event in HH_R1 is concerned with the energy consumption of the various appliances in HomeHub entity, while the event in FMFM_R1 is related to the creation of an exercise plan for the family. Examples of other types of requirements represented in RELAX for *FeedMe FeedMe* SoS can be found in [7].

B. Overlap Detection

As explained in [8], in order to identify conflicting requirements it is necessary to detect common aspects and elements shared by the involved requirements, also known as overlapping elements. This is an important activity during conflict management since requirements without overlapping elements cannot be considered as conflicting requirements [9].

In the *MaCoRe_SoS* framework, the overlap detection identifies requirements that share the same resources in an SoS. The framework uses the extension clause RESOURCE in the requirements specification, together with the shared ontology, to support overlap detection. The RESOURCE clause states which resources relate to each requirement. There may be requirements that do not use any resource or requirements that use more than one type of resource. The shared ontology is used to identify different representations of the same resource.

The overlap detection activity identifies intersections among the requirements and resources of the various entities, and provides a list of these intersections (possible sources of conflicting requirements), which is stored in the knowledge database, for future references, and to be monitored at runtime during conflict detection activity.

An example of overlapping requirements is found in HH_R1 and FMFM_R1 presented above. In this case, there is a potential for the requirements to be conflicting since both requirements are concerned with resource "electricity". This is an example of a CONSUME X PROTECT type of conflict.

C. Conflict Detection

In the conflict detection activity, the framework generates a set of assertions from the list of identified overlapping requirements. As the requirements are specified using RELAX, the framework can generate these assertions using fuzzy branching temporal logic (FBTL) [6], since the semantics of the RELAX expressions are defined in terms of FBTL [2]. The assertions are checked at runtime in order to identify the requirements conformity and possible conflicts. By using FBTL assertions, *MaCoRe_SoS* is able to handle challenges related to uncertainty in an SoS. An example of a FBTL assertion for the requirement HH_R1 is presented below.

RELAX Grammar Expression: SHALL (AS CLOSE AS POSSIBLE TO 100 *q*);
Formal FBTL expression: $AGF((\Delta(q) - 100) \in S)$
Definitions: *q* is "HomeHub control the home electricity usage"; *S* is a fuzzy set whose membership function has value 1 at zero ($m(0) = 1$) and it decreases continuously around zero; *AGF* are FBTL quantifiers.

The conflict detection activity is executed during runtime and is supported by an event monitor. The RELAX extension clause EVENT, in the requirements specification, maps the operations of a participating entity with their requirements. Once an entity performs an operation, an event notification is sent to the framework and the event monitor logs the event in the framework database. After an event is received, the framework identifies the requirements related to this event. Based on the identified requirements, the framework detects affected resources based on the information specified in the RESOURCE clause. The framework checks all the assertions related to those resources in order to identify if there is a violation in the FBTL assertion generated by the requirement specification. If a violation is detected, the framework reports that a conflict happened between the requirement that generated the event and the requirement that had its assertion violated.

As an example, consider requirement HH_R1 with the assertion presented above, and requirement FMFM_R1. Suppose events HomeHub-SaveEnergy and FMFM-FamilyExercisePlan are received by the framework, where the latter creates an exercise plan for the family that includes the use of several home appliances causing a higher electricy consumption. Suppose that after receiving the event FMFM-FamilyExercisePlan, the framework detects a violation in the resource electricity regarding the assertion for the requirement HH_R1 since the value of the home electricity usage rises above the expected value (> 100 KWh). In this case, a conflict involving HH_R1 is identified, due to resource violation caused by another requirement that uses the same resource. As the event that triggered the assertion violation was FMFM-FamilyExercisePlan, the framework identifies a conflict between FMFM_R1 and HH_R1.

V. PILOT STUDY

In order to provide an initial evaluation of the *MaCoRe_SoS* framework, we created a pilot study using a scenario based on the *FeedMe FeedMe* SoS. The scenario uses 16 requirements together with 14 associated events, for AnalyseMe entity (AM), HomeHub entity (HH), and *FeedMe FeedMe* SoS (FMFM). The requirements, resources, and events for each of these entities can be found in [7]. Using these requirements, we implemented a part of the *MaCoRe_SoS* framework as a set of simulated entities, which could be configured to raise specific events at different points in time, and report relevant values for the resources to which they were associated. The scenario used in this pilot study is presented in Table I.

The scenario presents the routine activities of a family of three members, and the arrival of a guest during the day. It presented three conflicts involving the requirements of the entities, the SoS, and the shared resources between them. For example, at time point 5 of the scenario, a conflict occurs because the operation of the appliance associated with the exercise proposed by AnalyseMe would result in a higher energy consumption, causing a conflict between AnalyseMe and HomeHub. The scenario also includes examples of conflicts involving requirements at the SoS level. For example, the arrival of a guest causes a conflict between the FMFM requirement to update the family meal plan, while at the same time it needs to maintain

the family food resources (time point 15). We ran the scenario with our initial implementation of MaCoRe_SoS. We monitored the utilization of the relevant resources, measured in arbitrary units for simplicity, and monitored the requirements assertions to identify conflicting requirements.

TABLE I. SCENARIO DESCRIPTION

Time	*Event*	*Scenario Description*
1	FMFM-MealPlan	FMFM creates a meal plan for breakfast.
2,3	HH-Monitoring	HomeHub initiates its operation by monitoring and controlling the appliances of the house.
4	FMFM-MealPlan	FMFM creates a meal plan for lunch.
5	AM-Exercises	AnalyseMe proposes some exercises to a family member **(Conflict 1 identified).**
6	AM-Monitoring	AnalyseMe checks the user's nutritional information.
7,8	HH-Monitoring	HomeHub keeps its operation by monitoring and controlling the appliances of the house.
9	FMFM-MealPlan	FMFM creates a family meal plan for dinner. (**Conflict 2 identified**).
10, 11	AM-Sleep Monitoring	The user's go to sleep and AnalyseMe monitors their sleep patterns.
12, 13	HH-Monitoring	HomeHub keeps its operation by monitoring and controlling the appliances of the house.
14	FMFM-MealPlan	FMFM creates a family meal plan for breakfast.
15	FMFM-GuestMealPlan Update	FMFM receives information about the arrival of an unexpected guest at home and updates the meal plan. (**Conflict 3 identified)**

A. Scenario

We created a simulated *FeedMe FeedMe* SoS environment, with three instances of the AnalyseMe entity (AM), one instance of the HomeHub entity (HH), and one instance of the *FeedMe FeedMe* SoS (FMFM), and submitted them to our implemented framework *MaCoRe_SoS*. The list of requirements of each entity can be found in [7]. During the overlap detection activity the framework identified a list of 24 overlapping elements involving 13 different requirements and five shared resources in the SoS environment. As explained in Section IV, the framework uses the extended clause RESOURCE, provided in the requirements specification, to identify resources shared by the entities. The list of the 24 detected overlaps was stored in the framework database and an assertion in FBTL was created for each one of the 13 requirements concerned with the identified overlaps. The assertions were monitored, in order to detect violations that may lead to conflicts, during execution of the various entities.

In this pilot study three conflicts were identified. The first conflict occurred at time point 5 between requirement AM_R4 (AnalyseMe proposes an exercise plan for a family member which requires the use of some home appliances) and HH_R1 (HomeHub reduction of electricity consumption). This conflict was identified after AnalyseMe instance raised event AM-Exercises. The framework received the event and identified that the event was related to requirement AM_R4. It used RESOURCE clause and identified resources CALORIES and ELECTRICITY. All the assertions involving resource ELECTRICITY (7) and resource CALORIES (3) were evaluated. The framework detected a violation in the FBTL assertion of requirement HH_R1 (as the electricity consumption went to 113 KWh and the maximum expected value were 100 KWh).

The second conflict happened between requirement FMFM_R2 (*FeedMe FeedMe* SoS proposes a family meal plan) and HH_R2 (HomeHub avoids excessive food consumption). Similarly, this conflict was identified, at time point 9, after *Feed Me Feed Me* instance raised event FMFM-MealPlan. The framework was able to identify that this event is related to requirement FMFM_R2 and to resources CALORIES, INSULIN, and FOOD. All assertions involving resources CALORIES (3), INSULIN (3) and FOOD (4) were evaluated. A violation was detected in the FBTL assertion of requirement HH_R2 (as the food resource reached 18 units which is bellow the minimum expected value of 20 units).

The third conflict happened between requirement FMFM_R4 (*FeedMe FeedMe* SoS proposes to adjust the meal plan after an unexpected guest arrives) and HH_R2 (HomeHub avoids excessive food consumption). This conflict was identified after *FeedMe FeedMe* instance raised event FMFM-GuestMealPlanUpdate (at time point 15). The framework was able to identify that this event is related to requirement FMFM_R4 resources CALORIES, INSULIN and FOOD. Similarly, all assertions involving resources CALORIES (3), INSULIN (3) and FOOD (4) were evaluated. The framework detected a violation in the FBTL assertion of requirement HH_R2 (as the food resource reached 13 units below the minimum expected value of 20 units).

B. Results and Discussion

The above pilot study shows the effect of conflict identification on the utilization of resources managed by the SoS using the framework. The results of the evaluation suggest that the framework is able to help an SoS to manage their resources by identifying conflicting requirements at runtime. A key limitation of our evaluation is that the results are based on a simulated environment of the *FeedMe FeedMe* scenario, of limited size and, therefore, it is not possible to claim generalizability at this stage. In addition, it is not possible to evaluate the scalability of the approach at this stage. Currently, we are analyzing other domains with realistic workloads to address these issues.

VI. RELATED WORK

Management of conflicting requirements in standalone software systems has been extensively studied in the literature [4][8]. In [8], the authors present a survey about the management of inconsistencies in software engineering composed of the following activities: detection of overlaps, detection of inconsistencies, diagnosis of inconsistencies, handling of inconsistencies, tracking of inconsistencies and specification and application of inconsistency management policies. Similarly, the MaCoRe_SoS framework includes three main steps: (1) conflict identification, with activities overlap detection and conflict detection; (2) conflict diagnosis, and (3) conflict resolution.

Prior work on conflict identification in standalone systems include logic-based approaches that use formal modelling languages to identify inconsistences in the models that may

lead to a conflict. These approaches use first-order logic [10], temporal logic [11], and Quasi-Classical (QC) logic [12]. All these techniques have potential to be applied to SoS environments, as long as the requirements are written in a formal logic-based language. Our framework uses fuzzy branching temporal logic to detect conflicting requirements. The goal-based works presented in [11] and [13] describe approaches to detect conflicting requirements at design time, while MaCoRe_SoS supports runtime detection of conflicting requirements.

Other approaches have been proposed to support identification of conflicting requirements based on model checking techniques [14][15][16][17][18]. The work in [14] presents an approach to analyse properties in requirements specifications written in SCR. Their results suggest that it is difficult to manage performance when using model checking. In [15], the authors present a model checking technique to detect unexpected emergent behaviour in a SoS. The work in [17] presents CML, a formal language for modelling SoS that supports model checking. However, as stated in [18], there are limitation on the number of behaviours that can be checked due to the size of CML models.

Overall, the proposed framework complements existing approaches and contributes to the area of consistency management by taking into account conflicts in the systems of systems environment.

VII. Conclusion and Future Work

The growth in the complexity and heterogeneity of modern systems has led to systems that compose themselves into bigger systems to achieve new functionalities. These systems are often System of Systems (SoS) where the management of emerging conflicting requirements is a challenge. In this paper we presented a framework called MaCoRe_SoS. The main goal of the framework is the management of resource-based conflicting requirements in SoSs. We have presented and discussed the first step of the framework, conflict identification based on two related activities: overlap detection and conflict detection. We built a prototype version of the framework and demonstrated its usage in a pilot study based on *FeedMe FeedMe*, an example SoS ecosystem designed to support food security. The results of the pilot study are promising, and demonstrate that it is possible to identify conflicts using the framework and support SoSs during resource management.

Currently, we are implementing the other steps in the framework (conflict diagnosis and resolution). We also plan to extend the framework to address conflicting requirements that are not only concerned with resources. Finally, we want to evaluate the framework in real-world SoS domains and analyze the performance of the framework under realistic workloads.

References

[1] M. W. Maier, 'Architecting principles for systems-of-systems', in *INCOSE International Symposium*, 1996.

[2] 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.

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

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

[5] A. Bennaceur *et al.*, 'Feed me, feed me: an exemplar for engineering adaptive software', in *Proceedings of the 11th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, 2016.

[6] 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.

[7] 'MaCoRe_SoS. http://sead1.open.ac.uk/macore_sos/'. .

[8] G. Spanoudakis and A. Zisman, 'Inconsistency management in software engineering: Survey and open research issues', *Handb. Softw. Eng. Knowl. Eng.*, 2001.

[9] G. Spanoudakis, A. Finkelstein, and D. Till, 'Overlaps in requirements engineering', *Autom. Softw. Eng.*, vol. 6, no. 2, pp. 171–198, 1999.

[10] B. Nuseibeh, J. Kramer, and A. Finkelstein, 'A framework for expressing the relationships between multiple views in requirements specification', *IEEE Trans. Softw. Eng.*, vol. 20, no. 10, pp. 760–773, 1994.

[11] A. Van Lamsweerde, R. Darimont, and E. Letier, 'Managing conflicts in goal-driven requirements engineering', *IEEE Trans. Softw. Eng.*, vol. 24, no. 11, pp. 908–926, 1998.

[12] A. Hunter and B. Nuseibeh, 'Managing inconsistent specifications: reasoning, analysis, and action', *ACM Trans. Softw. Eng. Methodol. TOSEM*, vol. 7, no. 4, pp. 335–367, 1998.

[13] R. Ali, F. Dalpiaz, and P. Giorgini, 'Reasoning with contextual requirements: Detecting inconsistency and conflicts', *Inf. Softw. Technol.*, vol. 55, no. 1, 2013.

[14] R. Bharadwaj and C. L. Heitmeyer, 'Model checking complete requirements specifications using abstraction', *Autom. Softw. Eng.*, vol. 6, no. 1, pp. 37–68, 1999.

[15] S. Malakuti, 'Detecting emergent interference in integration of multiple self-adaptive systems', in *Proceedings of the 2014 European Conference on Software Architecture Workshops*, 2014, p. 24.

[16] S. Malakuti, M. Aksit, and C. Bockisch, 'Runtime verification in distributed computing', *J. Converg.*, 2011.

[17] J. W. Coleman *et al.*, 'COMPASS tool vision for a system of systems collaborative development environment', in *System of Systems Engineering (SoSE), 2012 7th International Conference on*, 2012, pp. 451–456.

[18] J. Woodcock, A. Cavalcanti, J. Fitzgerald, P. Larsen, A. Miyazawa, and S. Perry, 'Features of CML: A formal modelling language for systems of systems', in *System of Systems Engineering (SoSE), 2012 7th International Conference on*, 2012, pp. 1–6.