Refining Missions to Architectures in Software-Intensive Systems-of-Systems

Eduardo Silva, Everton Cavalcante, Thais Batista
DIMAp – Department of Informatics and Applied Mathematics
UFRN – Federal University of Rio Grande do Norte
Natal, Brazil
eduasilva@gmail.com, everton@dimap.ufrn.br, thais@ufrnet.br

Abstract—An important concern in the design of a software-intensive system-of-systems (SoS) is modeling both global and individual missions to be accomplished by the SoS and its cooperating constituent systems. A first step towards the concretization of mission models is their refinement into architecture descriptions in terms of the constituents able to fulfill the established missions through emergent behaviors. In this context, we introduce M2Arch, a model-based process to refine mission models into architecture descriptions. M2Arch is concerned with the automatic generation of architecture descriptions in SoSADL, a formal language to describe SoS software architectures, from mission models in mKAOS, a language to model missions. M2Arch also comes with an associated tool supporting both mission modeling and architecture description as well as validation and simulation of the resulting architectures.

Keywords—systems-of-systems; missions; software architecture; architecture description language; model refinement

I. INTRODUCTION

Software-intensive systems-of-systems (SoS) exhibit distinguishing characteristics that have posed a set of challenges to the development of this new class of complex systems [1, 2]. SoS are evolutionary developed from independent, heterogeneous systems to achieve missions that cannot be accomplished by a single system working alone, thereby strongly relying on emergent behaviors resulted from interactions with other collaborating systems. Each constituent system has individual missions and may be able to contribute to the accomplishment of global missions of the overarching SoS, i.e., goals to be achieved or new functionalities to be offered by it.

Missions play a key role in the SoS context as they define required capabilities of constituent systems and the interactions among these systems that lead to emergent behaviors towards the accomplishment of the global goals of the SoS. Therefore, an important concern in the design of SoS is the systematic modeling of both global and individual missions, as well as all relevant mission-related information. To capture this information, missions can be represented as mission models and be used as a basis of the whole SoS development process, pervading activities such as architectural design, validation, verification, and simulation.

The literature still has lacks in terms of specific languages, models, and tools for defining missions in the SoS context [3]. Several approaches and tools were proposed for SoS modeling, but they do not handle mission modeling, an important part of SoS design. On the one hand, proposals for modeling missions in traditional systems neglect some important concerns such as emergent behavior, which is essential for the SoS context. On the other hand, the existing solutions for either SoS modeling or mission modeling are not fully applicable to missions in the SoS context as they require specific concepts for SoS and the notion of mission as a central element. To address these issues, our previous work [4, 5] has introduced mKAOS, a pioneering language aimed to support the specification of missions and the definition of relationships between such missions and other concerns of SoS. The main goal of mKAOS is to allow for a detailed modeling of missions in the SoS context and to enable stakeholders to identify or define specific elements for SoS, e.g., constituent systems, required capabilities, and/or desired emergent behaviors.

An SoS software architecture is recognized as a key factor for achieving missions. In this perspective, the next step towards the concretization of a mission model is its refinement to an architecture description, i.e., a model expressing the SoS software architecture while being compliant with the mission model. There are many challenges involved in SoS architecture description due to the inadequacy and lack of expressiveness of existing architecture description languages (ADLs) for this context [6, 7, 8]. Oquendo [9] points out two major weaknesses of these notations, namely (i) the impossibility of describing an architecture for which the architect does not know the concrete components at design-time and (ii) the lack of any concept or mechanism to cover emergent behaviors.

Inspired by requirements engineering solutions [10, 11, 12, 13], mission models can be refined to the architectural level to ensure traceability and ease maintenance. However, this process involves some additional challenges due to the inherent dynamism of SoS. First, it is not trivial to detect and validate the mission model, especially due to emergent behaviors. Second, mission models often do not deal with behavioral concerns and hence the validation of an architecture within the mission model cannot be statically performed. Third, it is highly desirable having a systematic process to support developers involved in activities such as mission modeling and architecture description of SoS.

Aiming at bridging missions and software architectures in SoS, our previous work [14] has briefly introduced a mapping...
mechanism of mission models expressed in mKAOS to architecture descriptions in SosADL [9], a formal, well-founded theoretically language targeting the description of SoS software architectures. SosADL concerns both structural and behavioral viewpoints of SoS software architectures while providing new concepts and language constructs to embody the features of software-intensive SoS. The major impetus of such a synergistic relationship between these models is the absence of behavioral descriptions in mKAOS and the lack of mechanisms to represent missions in architectural models. Mission and architectural models are complementary in this context and hence must be created and maintained to properly describe SoS.

The main goal of this work is to go a step further by introducing M2Arch, a model-based process to refine mission models in mKAOS towards the automatic generation of architecture descriptions in SosADL. M2Arch basically relies on a model-to-model (M2M) transformation [15] supported by an integrated tool for describing both mission models and architectural models. In addition, M2Arch comes with modeling tools sustaining the whole SoS design process, including validation and simulation of the resulting architectures.

This paper is structured as follows. Section II introduces the running example used along the paper as well as it presents both mission and architectural modeling languages. Section III presents the M2Arch process and the tool supporting it. Section IV discusses related work. Finally, Section V presents concluding remarks.

II. BACKGROUND

A. Running Example: A Flood Monitoring SoS

Floods are one of the major problems in many countries around the world. In rainy seasons, this type of event can be quite devastating in urban centers traversed by rivers as they may cause material, human, and economic losses. Regardless of their magnitude, floods represent a risk and hence they must be detected as quickly as possible. In this context, an SoS can foster effective flood monitoring, support timely response from authorities, and contribute to alleviate impacts caused by floods. Such an SoS can combine information provided by multiple collaborating independent systems such as river monitoring systems and meteorological systems. Within this SoS, river monitoring systems composed of a network of sensors spread in flood-prone areas near the river can be used to monitor the river water level as an indicator of flooding. In turn, meteorological systems comprising weather stations and satellites can be used to collect and analyze atmospheric parameters (e.g., temperature, humidity, rain amount and intensity, etc.) that also serve as input to the construction of prediction models supporting weather forecasting.

Despite these systems seem to be enough for enabling the SoS to determine the risk of a potential flood, false positives regarding a flood risk may be caused by biased sensors or other conditions on the river. Aiming at improving the accuracy of the measures collected by the sensor nodes deployed in the monitored river area, a surveillance system based on the remote use of drones can be used to provide images of the river for estimating its flow rate. In this scenario, drones endowed with digital cameras can be used to record video and/or capture images of the overflown area. These multimedia data are then processed and combined with data provided by the meteorological systems and data provided by the sensor nodes spread on the river, thus contributing to detect an imminent flood with maximum confidence and avoid false positives.

B. Mission Models in mKAOS

mKAOS [4, 5] is a specialization of KAOS [16] for SoS mission description in terms of six different models, each one handling a specific aspect of the SoS. The Mission Model is the main model, used to represent both individual and global mission and how they relate to each other. The Responsibility Model defines constituent systems that can assume responsibility over individual missions. The Object Model defines entities and events used by constituent systems. The Operational Capability Model defines operational capabilities of the constituent systems, i.e., functional operations performed by the system. The Communicational Capability Model represents the concept of communicational capabilities, i.e., the ability of constituent systems to cooperate with others. At last, the Emergent Behavior Model specifies emergent behaviors of the SoS and how these behaviors are related to the cooperation among constituent systems. mKAOS models can be overlapped to define relationships among elements from different contexts. For instance, it is possible to overlap the Mission Model and the Responsibility Model to assign responsibility of individual missions to constituent systems. Another possibility is overlapping the Responsibility Model and the Operational Capability Model to identify which capabilities can be associated to each constituent system.

Fig. 1 depicts the representation of the missions of the flood monitoring SoS resulted from the overlapping of a Mission Model and Responsibility Model. The Alert Citizen in Risky Areas mission is refined into two other missions, namely Identify Citizens in Risky Area and Alert Citizen. The first one is refined into two more missions, Calculate Risky Areas and Identify Citizen. The Identify Citizen and Alert Citizen individual missions are assigned to the Social Network System, while the Calculate Risky Area individual mission is assigned to the Surveillance System.

C. Architecture Descriptions in SosADL

To represent SoS software architectures, an ADL must consider: (i) both structural and behavioral definitions for the SoS and its constituent systems; (ii) interactions among constituent systems; (iii) adaptations due to the dynamic scenarios in which an SoS operate; and (iv) properties, constraints, and quality attributes [6]. To cope with these concerns, SosADL [9] arises as a formal language to describe SoS software architectures with support for rigorous analysis mechanisms. The formal foundations of SosADL rely on an extension of the π-calculus process algebra [17], thereby being a universal model of computation enhanced with SoS concerns.

SosADL uses the concept of abstract and concrete architectures in the sense that constituent systems and its communications will be realized at runtime. An abstract architecture
defines an SoS in terms of types of constituent systems, so that several concrete systems of a given type could be part of the SoS and form its concrete architecture. This is an important concern in the SoS context since the architect may be unable to predict which constituent systems will realize the SoS. Moreover, the internal architecture of these constituent systems is often unknown. For modeling purposes, we focus on abstract architectures since concrete ones can only be produced at runtime or in simulation environments.

SosADL uses a set of eleven elements to describe an SoS software architecture: (i) systems, the concept to represent constituent systems; (ii) gates, which represent the interfaces of the systems; (iii) connections, communication channels within a gate; (iv) assumptions, which represent properties expected by the gates and must be satisfied by the environment; (v) guarantees, which are properties that must be enforced by a constituent system; (vi) general properties that are relevant to the system; (vii) behavior, which represents how a constituent system implements some functionality; (viii) mediators, which are elements under the control of the SoS and are responsible for coordination and data exchange; (ix) duties, which represent the interface or contract of a mediator; (x) coalitions, which represent the set of constituent systems composing an SoS; and (xi) bindings, which define the topology of the SoS.

Fig. 2 shows a partial example of a constituent system described in SosADL. The Gateway system has a gate called notification, which is composed of two connections, measure (for receiving data) and alert (for sending data). The guarantee for this system defines a protocol stating that the gate receives values via the measure input connection and it sends values via the alert output connection. These actions are performed repeatedly, as expressed by the repeat construct.

Fig. 3 shows part of the textual description of a mediator in SosADL. The Replicator mediator is defined with a duty called replicate and a guarantee specifying that the mediator will receive a Parameter and simultaneously send it through both connections destination1 and destination2.

D. Mapping from mKAOS to SosADL

The mapping mechanism proposed in our previous work [14] automatically generates architectural templates in SosADL from mKAOS descriptions. This automatic generation is based on a model-to-model (M2M) transformation [15] to ensure traceability. However, these models deal with different aspects of the SoS and hence they must be maintained together during the whole development process.

The mapping mechanism relies on the common concept of both languages, the interfaces. In SosADL, an interface is represented by a gate, comprising a set of connections. In mKAOS, the interface is an implicit concept when overlapping an Operational Capability Model and an Object Model: each operational capability will use and produce data of a specific entity. The interface of a capability is the set of all required and produced data, whilst the interface of a system is the set of all interfaces of all capabilities of this system. After identifying the interfaces, the mapping mechanism ensures that each interface prescribed by mKAOS will have a corresponding one in SosADL.

Based on the interfaces, it is also possible to identify how the constituent systems will communicate with each other. This can be achieved by analyzing the Communicational Capability Model and identifying how data are exchanged.
among the collaborating constituent systems within the SoS. Communicational capabilities defined in the Communicational Capability Model will rely on some input/output data required/produced by some operational capability.

Knowing how operational capabilities produce/require data and the representation of how constituent systems will cooperate by exchanging these data, it is possible to identify how constituent systems will be connected to each other. Such a connection must be mediated due to the dynamic nature of SoS, so that the topology of the SoS in SosADL must use mediators defined upon communicational capabilities.

Any complete mission model able to sufficiently describe the SoS can be used to build the structural viewpoint regarding its architecture.

Fig. 4 depicts the mapping mechanism from mission models in mKAOS to architecture descriptions in SosADL. Step 1 is responsible for identifying the data types used in all communications and operations and generating their corresponding definition in SosADL. Step 2 consists of identifying and defining constituent systems. In Step 3, constituent systems are enhanced with gates based on the operational capabilities. Step 4 consists of defining mediators based on the communicational capabilities. Finally, Step 5 concerns the identification of the communications among constituent systems and the creation of the connections using mediators.

III. M2ARCH: AUTOMATED REFINEMENT OF MISSION MODELS TO ARCHITECTURE DESCRIPTIONS

By using the mapping mechanism presented in Section II.D, it is possible to produce an architecture whose elements can be easily traced to the mission models. However, the produced architecture description in SosADL covers only the structural viewpoint. To have a complete architecture, it is necessary to define the behavior of the elements, a concern not covered by mKAOS and that hampers its automatic generation to SosADL. Furthermore, a refinement methodology to produce architecture descriptions from mission models must encompass validation mechanisms to ensure that those produced architectures indeed realize the established missions. In this perspective, we have proposed M2Arch process that makes use of the mapping mechanism to automatically generate architecture descriptions in SosADL from mKAOS models, also encompassing validation and verification mechanisms to ensure that produced architectures meet the SoS missions.

A. The M2Arch Process

M2Arch consists of a three-step process, as illustrated in Fig. 5. The process starts with the definition of mission models in mKAOS. These models are provided as input for an automatic transformation mechanism that produces an architectural model to be enhanced by the architect with description under the behavioral viewpoint. The architecture is then executed in a simulation environment. The results of such a simulation must be validated within the Mission Model.
The second step regards validation. Validating software architectures consists in checking if an architecture meets its requirements, i.e., missions in the SoS context. The validation process usually is manual since the verification is somewhat subjective: the interpretation of the requirements depends on the architect. Specifically, M2Arch proposes using a simulation tool that will execute the architecture in a simulated environment. The architect can follow the execution of the architecture and identify how constituent systems behave and if the expected emergent behaviors arise. Moreover, it is necessary to identify possible undesirable behaviors and then modify the models to avoid them. This process is essentially manual and the architect must produce a set of adjustments to be done in both mission and architectural models. If the architect detects any adjustment to the mission model, then the process is re-started as it will have impact on the architecture. Nonetheless, some emergent behaviors may depend only on the behavioral description of the architecture and hence these changes must be directly applied to the architectural model.

In the third step, the validated architecture is provided to a verification tool, producing a verified architecture as result. The verification of SosADL models are naturally held by \( \pi \)-calculus. The architect must specify a set of assumptions and guarantees that will be verified at runtime or simulation. This mechanism can also be used to verify efficiency when comparing alternative architectures to identify the best one suited for its purposes. Currently, there is an SosADL project for simulation with DEVS [18], a simulation formalism that supports the dynamics of SosADL. The project is in an early stage, but it will be integrated with M2Arch process to support the validation and verification activities.

### B. Tool Support

As the mapping is implemented to be automatic, the steps in M2Arch are programmatically executed using a M2M [15] transformation. This ensures traceability of missions and simplify the architecture design process: the architect describes only the architecture behavior and details further elements not related to the mission model. To implement M2Arch, we rely on the existing implementations for mKAOS and SosADL, especially on the EMF [19] metamodels of both languages. As the tools for SosADL are still under development, our implementation is currently based on preliminary versions gently provided by its development team.

The implementation of M2Arch was developed using ATL [20], which was chosen due to two main reasons. First, the tools supporting both mKAOS and SosADL languages are based on the Eclipse environment [21], thereby easing their integration along with the ATL transformations towards producing a larger toolkit for SoS development. Second, ATL is often used by the model-driven development community, thus having consolidated tools and detailed documentation available. The main rules for the transformation from mKAOS to SosADL is partially presented in Fig. 6. The ProduceSos rule is responsible for controlling the whole transformation process, calling all other transformation rules. Following the steps depicted in Fig. 4, this rule generates datatypes from entities (Step 1), systems from constituent systems (Step 2), gates for the constituent systems (Step 3), and mediators from communicational capabilities (Step 4).
Although the transformation does not encompass all mKAOS elements neither the SosADL elements, it can still be realized in both directions. It is important to emphasize that both mission and architectural models are complementary to each other and must be independently maintained. In the M2Arch approach, we have chosen a constructive approach in which the refinement will produce a single architecture capable of achieving the required missions and emerge the desired behaviors. An alternative is to build a set of possible architectures and to verify the conformance of each one with the mission model, but this approach would be computationally expensive.

IV. RELATED WORK

Comprehensive Modeling for Advanced Systems of Systems (COMPASS) [22, 23] is a framework and methodology for building and maintaining SoS, coming up with a set of tools, methods, and formalisms to model and analyze SoS with an underlying formal notation. COMPASS concerns SoS modeling from requirements to architecture using SysML diagrams with extensions intended to provide a formal support for architectural descriptions. The architectural models are fully refined into CML (COMPASS Modeling Language), a formal executable language that allows for model simulation and analysis. In terms of requirement modeling, COMPASS takes advantage of traditional SysML requirements diagrams. The validation of the process is manual and consists of checking if the process complies with the specified requirements. On the other hand, COMPASS enhances SysML with CML code to model the architecture. The COMPASS approach is similar to ours in terms of proposing a well-defined architectural definition process for SoS. However, it uses the usual concept of requirements instead of missions, which represent a more adequate concept to the SoS context since it handles the dynamic nature of this class of systems [5]. In COMPASS, using requirements as starting point might lead to information loss, e.g., a requirement may be unable to properly represent priorities. Another difference is the mechanism used to produce architectures. COMPASS presents a set of guidelines to produce and validate architectures from requirement diagrams, but this process is mostly manual. In turn, our proposal encompasses an automated transformation that produces architecture descriptions using information described in the mKAOS models while ensuring traceability.

The approach proposed by Haley and Nuseibeh [24] aims at enhancing requirement models to obtain a more detailed, refined model. The main concern is allowing for a better understanding of the requirements towards software architecture descriptions. The process iterates over both architectural and requirements models, helping to understand the impact of requirements on the architecture as long it is built. The process is not sequential and the analyst can start at any step. It is necessary to: (i) define the behavior of existing systems; (ii) describe the architecture of existing systems using problem diagrams; (iii) describe the future, post-integration, SoS architecture; and (iv) describe the post-integration SoS behavior. After these steps, the approach proposes an analysis mechanism for correctness. To produce the architecture, the proposal suggests using the Twin Peaks model [25], which consists of gradually building the architecture together with requirements in a cyclic approach, thus enabling the architect to foresee the impact of a requirement in the architecture.

During the architectural design, the architect must identify the capabilities of the constituent systems and the required capabilities. To provide the required capabilities, the architecture must fulfill a set of assumptions to be verified in a final step of the architectural modeling. The proposal does not use a well-defined process to produce the architecture: the architect must build the architecture from requirements with no defined methodology or technique. On the other hand, M2Arch has a well-defined process to design the architecture. Even though the approach is similar to KAOS in terms of requirement engineering, the language used to architectural modeling is not an ADL and hence some software architecture concepts are not present.

V. FINAL REMARKS

In this work, we presented M2Arch, a model-based refinement process for SoS architectural modeling that uses missions as the starting point. This process includes a M2M transformation to generate SosADL abstract architectures based on mKAOS mission models. This architecture encompasses all the required constituent systems and mediators, also defining the topology of the system, i.e., how these elements communicate. We have also introduced the tool that implements this process, partially depicting its transformation rules.

Regarding future work, the most important one concerns the validation and verification activities. For validation purposes, we intend to implement a simulation mechanism for SosADL to enable the architect to observe emergent behaviors and validate them against the mission model. In turn, the verification is simpler as SosADL relies on formal mechanisms. The verification step may consist of testing a set of constraints to verify if the architecture fulfills certain properties. With both validation and verification, it is possible to compose a complete development environment for SoS software architecture, enabling architects to design and maintain the models, as well as to simulate and verify the produced architectures.

VI. ACKNOWLEDGMENTS

This work was partially supported by CNPq – Brazilian National Council for Scientific and Technological Development and CAPES – Brazilian Coordination for the Improvement of Higher Education Personnel. The authors also thank to Flavio Oquendo (IRISA-UMR CNRS/Université Bretagne Sud, France) for his valuable contribution to this work.

REFERENCES


