MDL: A Mission Description Language

Imane Cherfa[1] and Salah Sadou[2]

[1] University of Of Blida, Algeria
Imane.Cherfa@irisa.fr
[2] University of South Brittany and IRISA, France
Salah.Sadou@irisa.fr

Abstract

The concept of System of Systems (SoS) is widespread in the domain of system engineering. Its use in the field of software engineering serves to increase the level of reuse (from component to system). However, the volatility of the systems compared to components, calls into question the stability of the SoS's architecture. Indeed, the latter must perpetually adapt to changes in the environment of the SoS. Thus, it is no longer possible to rely on the architecture to manage changes required by the SoS. However, a SoS exists to fulfill a certain mission. And the definition of the latter is relatively stable compared to the architecture that implements the SoS.

We propose a Mission Description Language (MDL) that formalizes the definition of the mission. This way of defining the mission of a SoS has a twofold objective: provide engineers with a stable documentation of the SoS in order to manage its evolutions and give the ability to automatically generate its architecture. This last point also solves the problem of constantly adapting the architecture of the SoS to changes in its environment. The construction of MDL led us to collect concepts covering the notion of mission through different research fields. The meta-model of MDL represents the synthesis of the concepts we had found.

1 Introduction

To respond to the need of reducing the costs of the software development and maintenance, the principle of reuse has been widely used untill now. Several concepts have been used for this purpose, such as function, class and component. But with the abundance of systems in the life of every day, the temptation is great to reuse these existing systems to construct a new one. The latter is called System of systems (SoS). This will provide new possibilities, such as opportunistic construction of systems: build a system to meet an ephemeral need by reusing available systems at the time.

In the domain of system engineering, SoS is a concept that exists for a while and was widely used in the military application domain. In the latter, the construction of a SoS is guided by a strong concept: the mission. Thus, the SoS exists to meet the needs of a mission.

To achieve a mission, autonomous, distributed and heterogeneous subsystems chosen for their capabilities are used to interact during a time frame [7, 3]. This interaction in an open and non deterministic environments leads to the emergence of behaviours with wanted or unwanted properties that can affect or favour the fulfilment of the SoS's mission [3].

Let us consider the classical model of development for the construction of a SoS. In this classical approach, the architecture is considered as the backbone of the system and its most stable part. But with the dynamicity which is inherent to the environment of the SoS, the architecture should be automatically adaptable in order to enable service availability. This led several research works to target the dynamic reconfiguration of architectures.

We believe that the military vision of SoS is transposable to the software domain. Indeed, considering the environment of a SoS, mission becomes the most stable element. Therefore,

any changes must be based on the definition of the mission. In this case, a rigorous definition of the mission becomes a necessity. And with a formal definition of the mission architecture can be generated automatically. With this possibility, the problem of dynamic adaptation of the architecture will be reduced to a regeneration of the architecture.

To be able to describe a SoS through its mission, we need first to identify all the concepts, and their relationships, necessary to the mission definition. This is the work we had undertaken and this paper presents the obtained result. Thus, we propose a mission description language, called MDL. This language is presented through its meta-model (abstract language) that describes the involved concepts and their relationships.

The remainder of this paper is organized as follows: Section 2 introduces the collect of concepts related to SoS mission. Section 3 goes into detail of MDL by the presentation of its meta-model. Section 4 presents concluding remarks and directions for future research.

2 Concepts Related to Mission

In SoS domain, we noticed a lack of papers related to mission concept. Furthermore, there is no common base vocabulary for SoS mission, consequently no established definition. In return, we noticed that the mission concept appears in several domains under different definitions: set of goals to reach, set of activities or operations to be executed in a predefined order.

This observation was made after a literature review on domains easily transposable to that of the SoS: Software Engineering and System Engineering. The majority of the works concerned requirement and goal specification [10, 9, 15, 2, 14, 5] or behaviour specification [10, 9, 8, 2]. To collect concepts related to mission, we focused on the following fields from the above sited domains: System specification, Multi Agents Systems, Resilient Distributed Systems, Dynamically Adaptive Systems, Real-Time Systems, Architectural Frameworks. Applications from these fields share some characteristics with SoS as illustred in Figure 1.

The identified concepts are in bold in the text describing the language/framework where they are well defined.

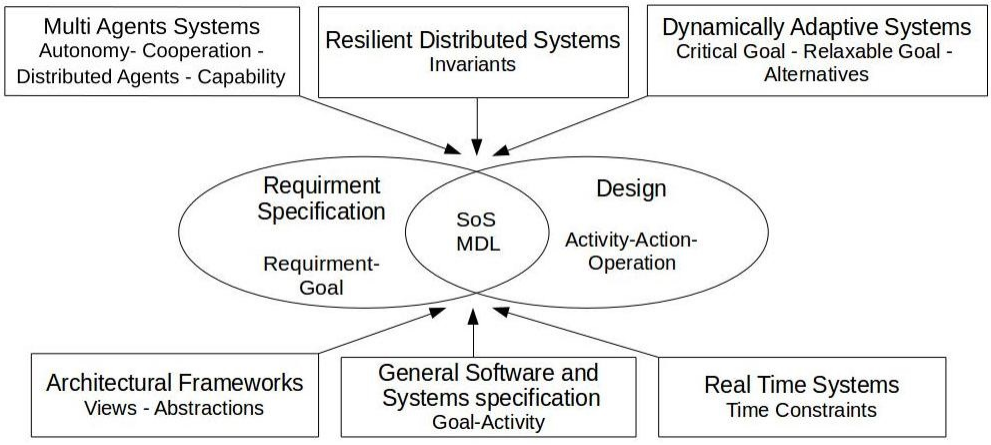

Figure 1: Context Of Mission Description Language

2.1 System specification

In any Software System, it is important to ensure that software satisfies stakeholders requirements. Thus many languages are dedicated to specify goals. As we are interested in describing SoS mission which is similar to the concept of goal, we analyze the most well known languages.

The Keep All Object Satisfied (KAOS) method [14] is a well established method for capturing and analyzing system requirements in form of goals, **assumptions**, and domain properties. The premise is to decompose the abstract high-level goals into more concrete sub-goals up to the level where goals represent requirements that can be handled by individual system agents.

The Unified Modelling Language (UML) [1] is the de jure standard industry language for specifying and designing software systems. UML addresses the modelling of requirments and activities by providing language constructs for describing uses cases, inclusions and extensions between uses cases, actors, **activities**, **interactions**, **sequencing** and **control flow of activities** .

2.2 Multi Agents Systems

Multi Agents Systems (MAS) share with SoS the caracteristic that sub-systems are often required to be autonomous. Moreover, like SoS subsystems, distributed agents have to cooperate with each other to achieve shared goals.

The Tropos methodology [2] is a similar approach to KAOS. Goals, **actors**, plans, **ressources**, **capabilities** and **dependencies** are modeled and analyzed from the perspective of the autonomous agents. In [12] authors extend the Tropos methodology to support high variability design by including alternatives, and revising the capability modeling. The authors aim at capturing both aspects of **ability** and **opportunity** to model the agent **capability**, based on the philosophical idea from [11] that can implies both ability and opportunity.

ADELFE [13] propose a methodology to develop adaptive MAS by focusing on **cooperative behaviour**. The main concept of ADELFE is the cooperative agent which has **skills, aptitudes, characteristics, communications** and a social attitude named **cooperation**.

2.3 Resilient Distributed Systems

Resilient Distributed Systems (RDS) share with SoS the characteristic of service availability while challenges occur.

The Invariant Refinement Method (IRM) [5] was proposed for systematical derivation of resilient distributed systems (RDS) architecture from high-level requirements. The main concepts supported by the method are: component, **invariant**, process invariant, change invariant, **assumption**, **knowledge dependency, knowledge flow, computation activity**.

2.4 Dynamically Adaptive Systems

In Dynamically Adaptive Systems (DASs) field, like in SoS one, self-adaptive systems have the ability to autonomously modify their behaviour at run-time to reach goals, when there is changes in their environment.

Relax [15] is a requirements language including explicit constructs for specifying and dealing with the **uncertainty** inherent in self-adaptive systems. It enables analysts to identify requirements that may be relaxed at run time when the environment changes.

The Software Component Ensemble Language (SCEL) [8] language is a kernel language that provides programmers with a set of linguistic abstractions for describing autonomic systems

in terms of Behaviors (processes, **activities** and targets), **Knowledge**, and Aggregations by complying with specific Policies.

2.5 Real-Time Systems

In Real-Time Systems (RTS) field, timing constraints is an important property for the accomplishment of the mission.

The UML Timed Sequence Diagram (TSD) [4] is an extension of UML Sequence Diagram to specify the real-time behaviour. TSD introduce **loops** and suggest a graphical notation. In [13] the author consider Timed Sequence Diagrams as requirement specifications which should be satisfied by other dynamic models of the system that are closer to the implementation.

The Timed Property Sequence Chart (TPSC) [6, 16] is an extended graphical notation of a subset of UML 2.0 sequence diagrams. TPSC provides a completely graphical front-end for software designers. It include timed constructs to specify **timing properties** of real-time systems. The main concepts holded by this laguage are: component instances, messages, **constraints, Clocks** and operators (**loop, synchronisation**).

2.6 Architectural Frameworks

Architecture frameworks provide a roadmap for describing the architecture of a system. This descriptions are necessarily done from multiple view-points. we selected those that are more suited to SoS.

The Department of Defense Architecture Framework (DoDAF) [10] is an architecture framework for the United States Department of Defence. In the DoDAF framework, there is several views, each of which is broken down into products and data: operational view, systems and services view, etc. The **mission** is described by a concept of operations and is organized by **capabilities**. capabilities are described by threads, and threads are described by **activities** executed in **serial or parallel**.

The Ministry of Defence Architecture Framework (MoDAF) [9] is an architecture framework for the UK Ministry of Defence. Similar to DoDAF, MoDAF provides a set of views that provide a standard notation to capture information about a business in order to identify ways to improve it. There is several views: strategic, operational, service orientated, etc. The main concepts supported by MoDAF meta-model are: **mission, capability, operational activity, capability constraints, capability dependency**, task, location and process.

3 MDL meta-model

The concepts identified in the previous Section are presented, in this Section, in an adequate manner with their relationships. In order to make our language evolutive, we defined its abstract syntax as a meta-model. In the following, we describe the core of MDL in more detail.For reason of clarity, we define two sub-meta-models, which share the mission concept that altogether form the MDL meta-model: i) the mission within the System of Systems part contains the main building blocks of a SoS and thus includes concepts like subsystems, Mission, Synergism, Capability, Role, or Environment. ii) the second part concerns concepts related to the refinement of the mission.

3.1 Mission within the System of Systems

The sub-meta-model given in Figure 2 contains the main building blocks of a SoS. Hardware/-Software systems, or humans are considered as *subsystems*. A *subsystem* can cooperate with another one. A set of *cooperation* leads to a *synergism* that is represented as several behaviours.

Mission accomplishment can be favoured by *desirable behaviour*, or negatively affected by *undesirable behaviour*, resulting from *cooperation*. It can be also affected by *environment assumptions*.

A *mission* is considered as part of *Role*. The concept of *Role* is needed to represent set of *capabilities* of the same context assigned to one or several *subsystems*. Each subsystem has one or more capability that is related to its ability to do something, and to the existence or no of the *opportunity* to do it. By the concept of *opportunity* we mean that the environment assumptions are sufficient to the success of the *mission* accomplishment.

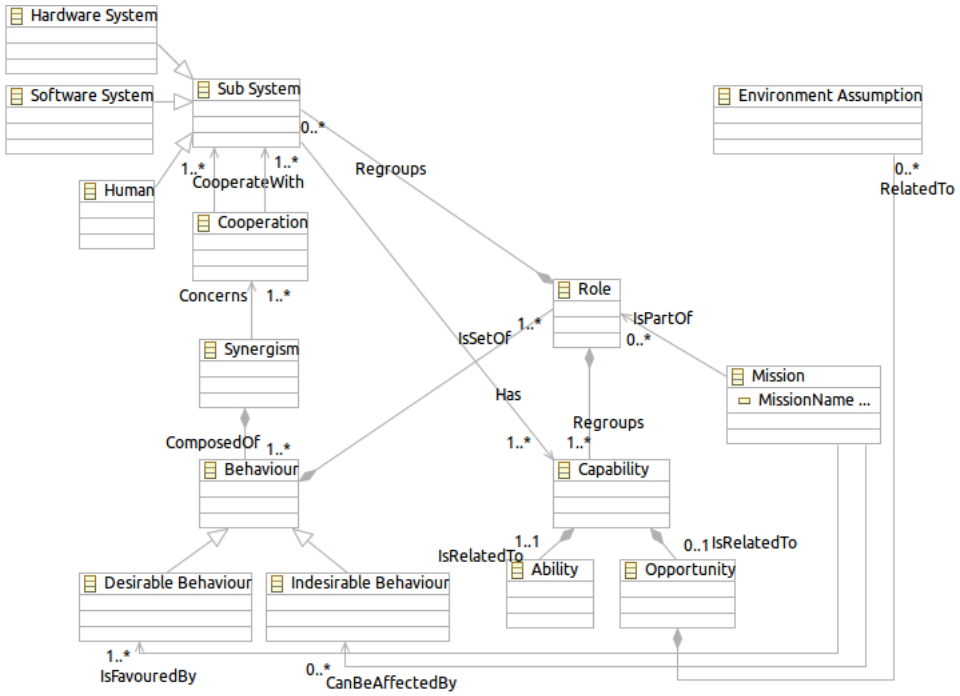

Figure 2: MDL meta-model

3.2 The Mission refinement

The part of the MDL meta-model concerning the *mission* refinement is shown in Figure 3. There is two sub-metaclasses of *mission* metaclass. A *critical mission* represent the mission that must be achieved even if there is an emergency situation while a *mission* can be temporary delayed.

A *trigger* is an event that launch the execution of the *mission*, if a *condition* is verified under a *timing constraints*. The existence of a *condition* or *timing constraints* is not mandatory.

A mission may be refined into submissions using *decomposition (AND/OR/XOR)*. To define a mission, the *decomposition* are not sufficient. We added *operators* allowing execution ordering

between submissions. These *operators* are either *simple* or *composite*. The *Loop* operator means that a mission execution must reiterate. The *Synchronization* operator signify that there is one or more missions that must be achieved before executing the current mission. The *Parallel* operator imply that the missions have to be executed in parallel. The *Sequential* operator means that the missions have to be executed sequentially. The *Neutralisation* operator indicate that the achievement of one mission imply the abandonment of the specified mission. The *Conflict* operator is used when a mission may negatively affect another mission.

Finally, a *mission* is associated to an *activity* satisfying a *precondition*, maintaining an *invariant* in the execution, and comprising *operations*.

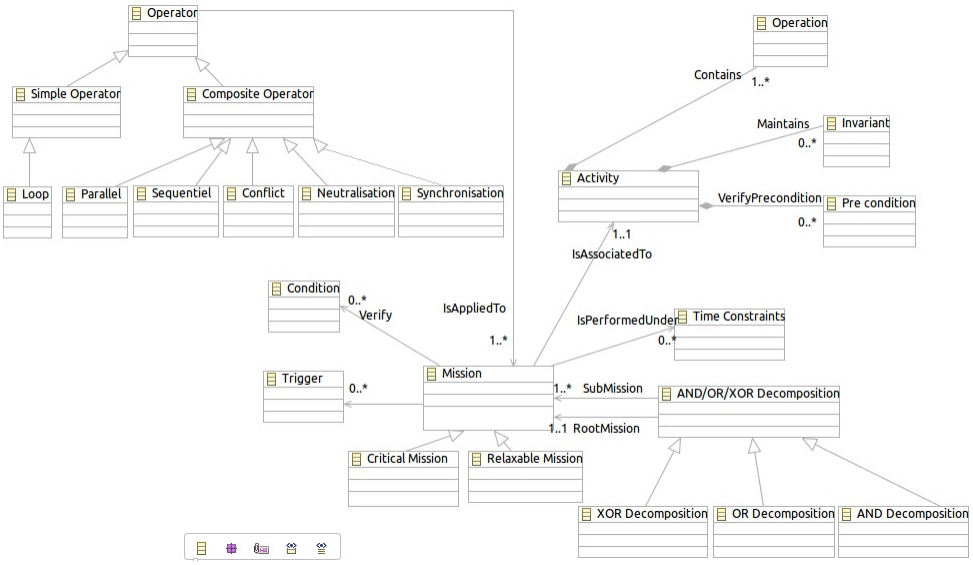

Figure 3: MDL meta-model

4 Conclusion

This paper presents a Mission description Language that defines an abstract syntax for modelling the mission. This language has a twofold objective: provide engineers with a stable documentation of the SoS in order to manage its evolutions and give the ability to automatically generate its architecture. This last point also solves the problem of constantly adapting the architecture of the SoS to changes in its environment.

Our future work concerns the definition of a formal semantic for MDL. This will allow us to check some properties like timing. This will also allow us to verify, validate a mission description and then automatically generate its corresponding architecture.

References

[1] Grady Booch, James Rumbaugh, and Ivar Jacobson. *Unified Modeling Language User Guide, The (2Nd Edition) (Addison-Wesley Object Technology Series)*. Addison-Wesley Professional, 2005.

[2] Paolo Bresciani, Anna Perini, Paolo Giorgini, Fausto Giunchiglia, and John Mylopoulos. Tropos: An agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3):203–236, may 2004.

[3] Vincent Chapurlat and Nicolas Daclin. *Risks and Resilience of Collaborative Networks: 16th IFIP WG 5.5 Working Conference on Virtual Enterprises, PRO-VE 2015, Albi, France,, October 5-7, 2015, Proceedings*, chapter System of Systems Architecting: A Behavioural and Properties Based Approach for SoS "–ilities" Modelling and Analysis, pages 591–603. Springer International Publishing, Cham, 2015.

[4] Thomas Firley, Michaela Huhn, Karsten Diethers, Thomas Gehrke, and Ursula Goltz. Timed sequence diagrams and tool-based analysis: A case study. In *Proceedings of the 2Nd International Conference on The Unified Modeling Language: Beyond the Standard*, UML'99, pages 645–660, Berlin, Heidelberg, 1999. Springer-Verlag.

[5] Jaroslav Keznikl, Tomas Bures, Frantisek Plasil, Ilias Gerostathopoulos, Petr Hnetynka, and Nicklas Hoch. Design of ensemble-based component systems by invariant refinement. In *Proceedings of the 16th International ACM Sigsoft Symposium on Component-based Software Engineering*, CBSE '13, pages 91–100, New York, NY, USA, 2013. ACM.

[6] W. Li and P. Zhang. On the semantics of scenario-based specification based on timed computational tree logic. In *2013 22nd Australian Software Engineering Conference*, pages 1–10, June 2013.

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

[8] Rocco De Nicola, Michele Loreti, Rosario Pugliese, and Francesco Tiezzi. A formal approach to autonomic systems programming: The scel language. *ACM Trans. Auton. Adapt. Syst.*, 9(2):7:1–7:29, July 2014.

[9] UK Ministry of Defence. Modaf. gov.uk/guidance/mod-architecture-framework.

[10] U.S. Department of Defense. Dodaf. dodcio.defense.gov/Portals/0/Documents/DODAF.

[11] Lin Padgham and Patrick Lambrix. Formalisations of capabilities for bdi-agents. *Autonomous Agents and Multi-Agent Systems*, 10(3):249–271, 2005.

[12] Loris Penserini, Anna Perini, Angelo Susi, and John Mylopoulos. High variability design for software agents: Extending tropos. *ACM Trans. Auton. Adapt. Syst.*, 2(4), november 2007.

[13] Gauthier Picard and Marie-Pierre Gleizes. *Methodologies and Software Engineering for Agent Systems: The Agent-Oriented Software Engineering Handbook*, chapter The ADELFE Methodology, pages 157–175. Springer US, Boston, MA, 2004.

[14] Axel van Lamsweerde. Requirements engineering: From craft to discipline. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, SIGSOFT '08/FSE-16, pages 238–249, New York, NY, USA, 2008. ACM.

[15] J. Whittle, P. Sawyer, N. Bencomo, B. H. C. Cheng, and J. M. Bruel. Relax: Incorporating uncertainty into the specification of self-adaptive systems. In *2009 17th IEEE International Requirements Engineering Conference*, pages 79–88, Aug 2009.

[16] P. Zhang, B. Li, and M. Sun. A timed extension of property sequence chart. In *High Assurance Systems Engineering Symposium, 2008. HASE 2008. 11th IEEE*, pages 197–206, Dec 2008.