# Model-Based Development of Systems-of-Systems with Reliability Requirements

Imad Sanduka
Airbus Group Innovations
81663 Munich, Germany
imad.sanduka@airbus.com

Roman Obermaisser
University of Siegen
Hölderlinstr 3, 57068, Siegen, Germany
roman.obermaisser@uni-siegen

***Abstract* — The development of Systems-of-Systems (SoS) architectures is challenged by the inherent characteristics of SoS such as operational independence, heterogeneity of constituent systems, emergent behavior and large-scale distribution. At present, the resulting complexity restricts the design space exploration of SoS architectures to be focused on cost and functionality. In this paper we extend our previous work on timing analysis and optimization in early SoS design phases by supporting reliability requirements in the generated SoS architecture. Our approach defines a SoS architecture development methodology that applies an architecture optimization method based on concise modeling with architecture patterns, timing and reliability requirements, and extensions to the Unified Profile for DoDAF and MODAF (UPDM). Optimization using Mixed Integer Linear Programming (MILP) is used to satisfy the real-time and reliability requirements and optimization results are back annotated to UPDM models.**

***Index Terms*—architecture optimization, architecture patterns, system reliability, reliability engineering, constituent systems, system of systems, UPDM model**

## I. Introduction

Model-Based System Engineering (MBSE) methodologies [1] become a widely used approach for system design and development. They organize the system development process by documenting and formalizing it using a set of models that facilitate communication between system stakeholders at different development stages. MBSE covers system engineering from system requirements analysis down to system verification and validation. It connects these stages and also establishes traceability of changes during the development process.

In MBSE modelling languages are used to capture different aspects of the system. The Unified Modelling Language (UML) [2] and System Modelling Language (SysML) [3] are well-known modelling languages in MBSE. However, using these languages for modelling large-scale complex systems such as Systems-of-Systems (SoS) with a wide variety of constituent systems results in complicated and huge diagrams. The models are typically difficult to read and follow for system developers and they can hardly be interpreted by customers. A major challenge is the integration and interoperability of constituent systems due to conflicting goals and heterogeneous constituent-system interfaces.

SoS are constructed out of autonomous constituents systems that are characterized by operational and managerial independence, geographical distribution and evolutionary development [4]. Describing these systems with their attributes, behaviors, and extra-functional properties in a model is challenging due to system complexity and the large community of different constituent-system owners that increase the number of SoS stakeholders. Architecture modeling frameworks were introduced to overcome these SoS modelling challenges. These frameworks establish views of the SoS from various perspectives (e.g., operational, functional) and they distinguish different levels of abstraction. The US Department of Defense Architecture Framework (DoDAF) [5], British Ministry of Defense Architecture Framework (MODAF) [6] and NATO Architecture Framework (NAF) [7] are the mainly used architecture frameworks for SoS modelling. The Unified Profile for DoDAF and MODAF (UPDM) [8] is a modeling language that was developed by the Object Management Group (OMG) group to support these profiles. It extends the UML and SysML profiles with specialized stereotypes and diagrams to support the architecture framework elements.

Synthesizing a system architecture that satisfies the system requirements is a major phase in the system development process [9]. At this stage the constituent systems of the SoS are defined with their relationships, parameters and interactions. The wide variety of constituent systems with respect to their functionality, quality, cost and technical properties increases the complexity in generating the system architecture and extends the search space. It is required that the constructed system architecture satisfies the functional and extra-functional requirements such as safety, reliability and security. Design decisions include the selection of the types of constituent systems along with their parameters and connections. The establishment of the system architecture involves an optimization with respect to the design goals and cost under the specified constraints.

In the field of architecture optimization IBM presented a generic approach called concise modelling [10], which provides a generic methodology for system architecture optimization. However, the approach is defined at the level of monolithic systems and lacks a clear connection to non-

978-1-4799-6649-3/15/$31.00 ©2015 IEEE

functional requirements. In previous work, we used concise modeling for supporting real-time requirements in SoS [11]. In this paper we extend the concise modeling approach to support system reliability in the optimization process of the SoS architecture development.

Several optimization frameworks for system reliability were introduced in previous work. For example, in [12] various optimization techniques are presented for solving system reliability problems where the element reliability is assumed to be fixed and the redundancy degrees are to be determined. Other techniques that deal with optimizing the component reliability and the redundancy degrees are described in [13] and [14].

Reliability in SoS as addressed in this paper is an open research problem. Considering system reliability in the development process becomes more complex at the level of large and complex systems such as SoS. The characteristics of the SoS result in challenges during SoS design where emergent behavior, an evolutionary development process, and an increased state space needed to be considered. In particular, these challenges are unsolved when moving form directed SoS, where the system has central control, towards virtual SoS with no central management and no common purpose of the constituent systems. The complexity remains unsolved in modeling and optimizing the SoS reliability using architecture frameworks. The current frameworks lack the methodology in defining and reflecting the SoS reliability at different modeling levels, and to include the constituent systems reliability properties in the constructed or generated SoS architecture.

We apply two means for enhancing system reliability: fault avoidance and fault tolerance [15]. The goal of fault avoidance is to carefully select the type and the quality of the chosen constituent systems during the development process in order to prevent later constituent-system failures at run time. Fault tolerance aims at containing and masking failures of individual constituent systems at run time, thereby limiting their effect on the services of the SoS.

In our previous work [11] we introduced temporal requirements in SoS architecture generation. In this paper we extend the previous methodology by adding the reliability requirements to the considered specifications of the generated SoS architecture while preserving the real-time specifications.

The remainder of the paper is structured as follows. Section II is an overview of SoS reliability engineering and the reliability block diagram. Section III describes the reliability formulation processes in architecture models. Reliability optimization and process integration with architecture patterns are discussed in Section IV. Section V presents the reliability analysis in UPDM with the proposed methodology and the development steps. The methodology and the results are discussed based on an example in Section VI. The paper ends with a discussion and conclusion.

## II. SoS Reliability Engineering:

Reliability is a Key Performance Factor (KPF) in building the SoS architecture and in the acquisition of its constituent systems [16]. SoS are widely used in critical applications such as military, emergency response, airports and water management, where a system failure can cause a substantial damage and affect human lives.

Modelling, analyzing and optimizing system reliability properties in early design phases of the SoS architecting process avoids costly change efforts in later phases where the SoS architecture fails to meet its reliability requirements. System reliability can be improved by fault avoidance including the enhancing of the quality of constituent systems, reducing the system complexity and practicing a planned maintenance and repair schedule [12]. In addition, system reliability can be increased by fault tolerance such as active redundancy or fault reconfiguration by the reassignment of failed services [17].

Reliability engineering is the discipline where methods and tools are developed to support the system design process in building reliable and maintainable systems by providing techniques for predicting, evaluating and demonstrating system reliability and maintainability [18]. Reliability requirements are considered as extra-functional requirements that assure the system's ability to perform the correct services with a high probability. The target value of system reliability is determined according to the system requirements. During the design process these reliability values are included in the optimization and system development constraints.

Reliability analysis becomes more complex at the level of SoS where a malfunction in any of its constituent systems may lead to an SoS failure. We regard each constituent system of the SoS as a fault containment region, which is independent from other constituent systems with respect to the immediate impact of a fault. Thus, at design phase, it is important to analyze the constituent-system reliability and the effect of these systems on the overall SoS reliability to build a Reliability Block Diagram (RBD) [18]. An RBD is an event diagram that provides developers with the information about reliability dependencies within the systems. It can provide insight about constituent systems that affect the functionality of the SoS and it provides information about the effects of a constituent-system failure on the overall SoS.

As shown in Fig. **1** the operational analysis of the SoS is the starting point for defining the SoS reliability diagram. An SoS operational activity is a group of functions that work together for achieving a certain SoS task. This high level of abstraction can also be conducted and understood by SoS customers for building the SoS operational scenarios that define the SoS purpose. The required SoS operations and their flow are defined in the operational activity diagram. From the operational level we move to lower levels of abstraction by describing the required SoS functions for implementing each of the listed operational activities. We map these functions to the operational activities in order to establish the functional flow diagram. Each function is associated with a constituent system

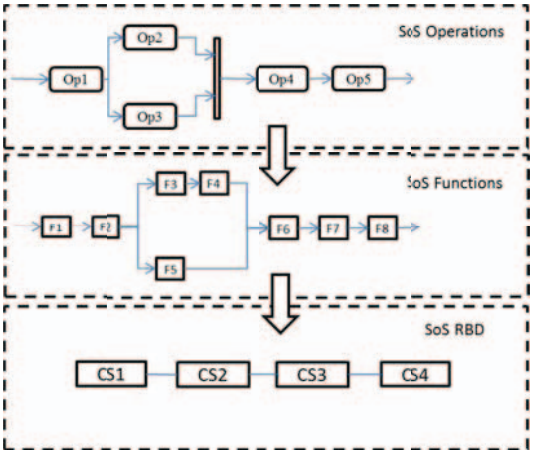

Fig. 1 Procedure for Defining SoS Reliability Diagram

that implements the function. By mapping functions to the constituent systems we refine the SoS reliability calculations from the SoS operations to the SoS constituent systems. Thereby, the SoS reliability diagram is created, which describes the SoS constituent systems and their reliability dependencies according to the functional flow diagram and functional dependencies.

## III. Reliability Formulation in Architecture Models

From the point of view of the dependencies between the components different system structures can be distinguished for reliability optimization [17] such as:

*1) Parallel–series systems:* Components are connected in series and redundant components are added in parallel.

*2) General network systems:* A complex system configuration with bridge networks and non-series non-parallel structures is used.

In this paper we cover the first structure for series systems with redundant constituent systems added in parallel to enhance the system reliability.

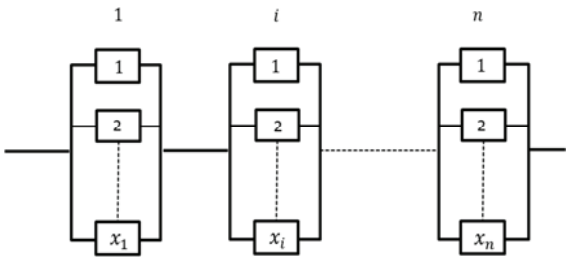

Fig. 2 Parallel-Series Reliability Structure

As shown in Fig. 2, a SoS consists of multiple constituent systems that are connected in series with redundant constituent systems in parallel. It is assumed that the failure mode of the constituent systems is fail-silent (e.g., established by self-checking and local diagnosis) [19]. If $R_i$ is the reliability of i-th constituent system with $0 \leq R_i \leq 1$ and $x_i$ expresses the redundancy degree for the i-th constituent system with $x_i \geq 1$, then the systems reliability $R_s$ is [20]:

$$R_s = \prod_{i=1}^{n}[1-(1-R_i)^{x_i}] \quad (1)$$

The optimization model of the systems is:

$$\begin{aligned} &\max && \prod_{i=1}^{n}[1-(1-R_i)^{x_i}] \\ &s.t. && \\ & && g(x_i) \leq b \\ & && x_i \geq 1 \quad \forall i \in \{1,\dots,n\} \end{aligned} \quad (2)$$

Where $g(x_i)$ are the optimization constraints and b is an integer value.

This kind of optimization problem is a nonlinear integer programming problem and algorithms typically cannot guarantee convergence to the global optimum within reasonable time. The goal of our modeling approach is to facilitate the problem formulation for the system architect in an efficient manner. For this purpose, we reduce the complexity of the optimization problem formulation. We can reformulate the problem by converting it into a linear optimization problem by two steps:

*1) Use of logarithm:*

$$\begin{aligned} &\max && \sum_{i=1}^{n} \ln(1-(1-R_i)^{x_i}) \\ &s.t. && \\ & && \ln g(x_i) \leq \ln b \\ & && b \neq 0 \end{aligned} \quad (3)$$

*2) Pre-calculated values:* The values of $(1-(1-R_i)^{x_i})$ can be provided as discrete values using external calculation tools. The output of the optimization process includes the constituent-system types and their redundancy degree. In concise modelling the input catalogue for the optimization process is provided through a spreadsheet, were the instances of the constituent-system classes and their property values are listed. We consider the redundancy degree as one of the properties of the system and the reliability for each redundancy degree will be calculated in the spreadsheet and provided as a discrete value.

## IV. Reliability Optimization and Architecture Patterns

In our reliability analysis approach we start by building the RBD. We formulate the optimization problem based on the RBD and we construct the corresponding architecture patterns that will be used in the optimization process (cf. Fig. 3).

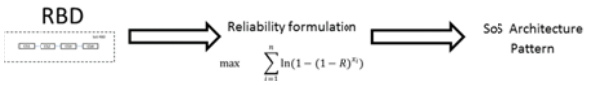

Fig. 3 Reliability optimization and architecture patterns processing steps

An architecture pattern denotes a set of constituent-system classes and their interconnection to realize a particular function [21]. It is guidance for the optimization process on how to connect the constituent systems and their distribution in the SoS architecture. It also indicates the purpose of the architecture and the flexibility of choosing the constituent systems within the architecture constraints. The architecture patterns facilitate SoS evolution to meet new requirements and enhance the SoS architecture reusability when modeling new systems with the same functionality.

In our work we use architecture patterns for SoS reliability optimizations. The architecture pattern is built to allow reliability enhancements of the system either by fault avoidance (i.e., constituent system selection), fault tolerance (i.e., redundancy degree) or both. In fault avoidance we manipulate the quality and the type of the constituent systems according to the available constituent systems catalogue. In this approach the architecture pattern is built out of constituent-system classes that allow the developer to optimize which constituent-system types best match the system's reliability requirements. In fault-tolerance patterns the architecture pattern allows for system redundancy. It provides the system architect with the ability to optimize the redundancy degree for each of the constituent systems as required to achieve the target reliability. It is also possible to use patterns that consider both fault avoidance and fault tolerance in the reliability optimization process.

Architecture patterns can also consider further requirements of the system such as system cost and real-time requirements. In [22] it was presented how architecture patterns can be used in SoS architecture optimization.

## V. Reliability in UPDM and Architecture Optimization Process

This section describes the development process of a reliable SoS. The development process is structured into the following steps (see Fig. **4**):

*1) SoS operational activities:* SoS operational scenarios are analysed according to the system requirements. Discussions between the SoS developer and the customer result in the operational activities flow diagram. The UPDM operational activity view is used for this purpose where the operational activities are listed and connected at a high level of abstraction that allows a common understanding between system architects and customers. At the operational level the customer demands a certain reliability for each scenario or operation. The required reliability is then mapped to the operational activities and considered as a reliability constraint for the design.

*2) Operational activities to functional mapping:* For each of the operational activities listed in the aforementioned diagram, the functions assigned to fulfil the activity are specified and mapped to the activity. At this stage the development process is moved from the customer domain to the developer domain. The mapping process is implemented using the "Operational Activity to Systems Function Traceability Matrix" of UPDM.

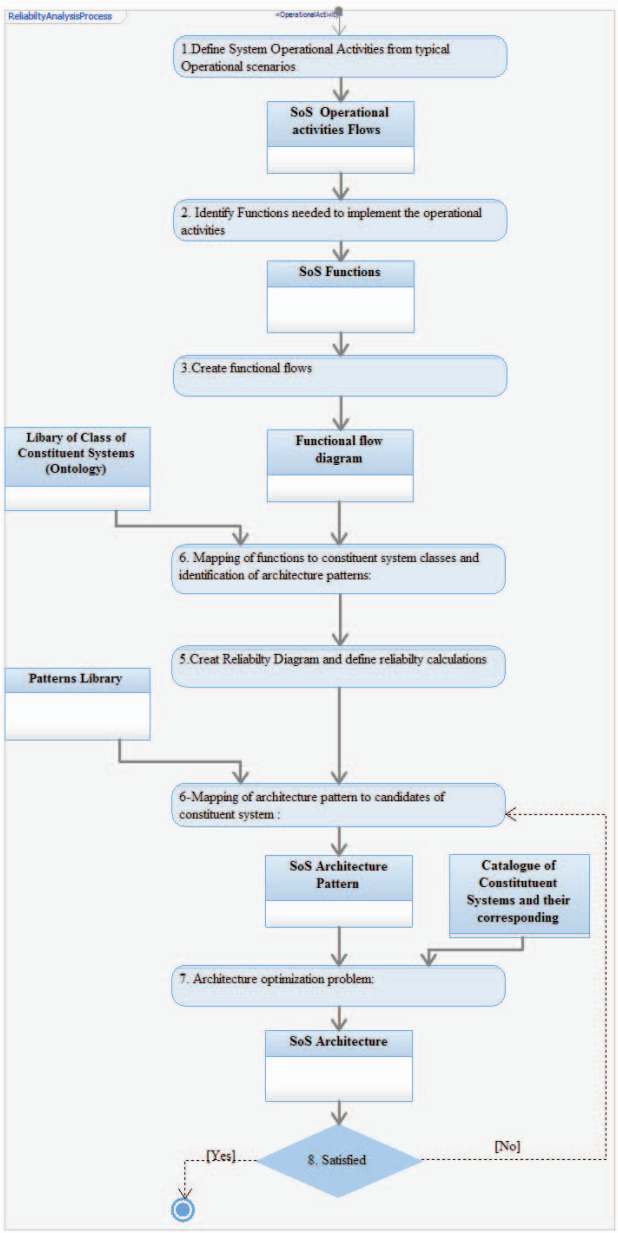


Fig. 4 Methodology Steps

*3) Functional Flow Diagram:* Based on the operational activities to the functions mapping process, the functional flow diagram is created with the "System Functionality Description" diagram of UPDM. The functional flow diagram contains all the functions required for the SoS operations, connected according to their precedence and dependencies.

*4) Mapping of functions to constituent Systems and identification of architecture patterns:* The functions mentioned in the previous step are mapped to the constituent-system classes that handle these functions. A class is defined at a high level of abstraction without any specification of concrete instances. The mapping process is implemented using the "System Interface Description" diagram of UPDM. This process guides the creation of architecture patterns, each

denoting a set of constituent-system classes and their interconnection to realize a particular function.

*5) Create reliability block diagram:* using the functional flow and functions to constituent-systems mapping process as a guideline, the reliability block diagram (as shown in Fig. **1**) is created according to the criteria described in section II.

*6) Mapping of architecture pattern to candidates of constituent systems:* The architecture patterns are either created or imported from a pattern library. At this step the candidates for the constituent-system classes are added to the architecture pattern and denoted as the constituent-systems catalogue.

*7) Architecture optimization problem:* This is the step where the optimization criteria are defined, i.e., optimization goals and constraints. The algebraic equations for goal and constraint calculations are derived and attached to the properties of the system. Examples for optimization goals are minimum cost and minimum latencies [11]. In this paper we include the system reliability as an optimization goal and as constraints. The calculations for system reliability are derived from the reliability block diagram.

*8) SoS Architecture:* The result of the optimization process is the SoS architecture. The optimization process specifies which system is better according to the optimization goals and constraints. The architecture pattern indicates the connections and interaction roles between the constituent systems. Optimization results are back annotated to the system model in the UPDM "System View" with concrete constituent systems and parameter values. If the results are not within the user expectations, steps 6 and 7 are repeated while changing one or a combination of the following items:

- Architecture patterns
- Optimization goals
- Optimization constraints
- Constituent-systems catalogue

## VI. Case Study

In our case study we consider a parallel-series SoS with redundancy. The purpose of the process is to optimize the selection of constituent systems and to create the SoS architecture that fulfils the system requirements with minimum cost and maximum reliability.

In Fig. 5 the system pattern is illustrated, where A,B,C,D are constituent-system classes connected in series. The problem is to decide which instances of the constituent systems A,B,C and D to choose from the constituent-system catalogue and to determine the redundancy degree for the constituent systems B and C that achieve the required SoS reliability within the minimum SoS cost.

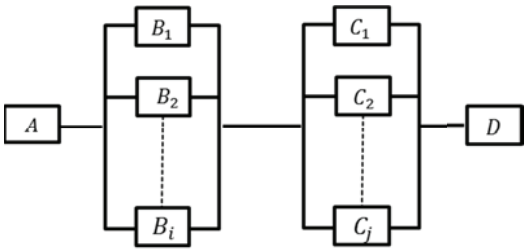

Fig. 5 Example Architecture

The problem is solved by formulating a multi-objective optimization problem:

$$\begin{aligned} &\max \quad R_s = \sum_{i=1}^{n} \ln(1-(1-R_i)^{x_i}) \\ &\min \quad g = \sum_{i=1}^{n} c_i x_i \\ &s.t. \qquad\qquad (4) \\ &\qquad g < g_0 \end{aligned}$$

Where g is the SoS cost function, $c_i$ is the cost of constituent system i, and $g_0$ is the cost constraint.

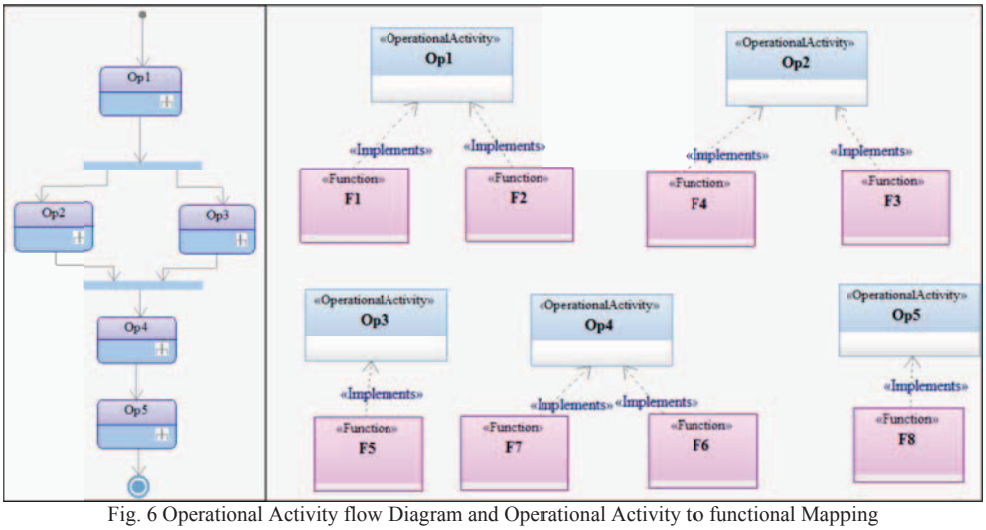

Fig. 6 Operational Activity flow Diagram and Operational Activity to functional Mapping

As discussed in section V the starting point of the model is the SoS operational analysis. In Fig. 6 the SoS operational activities are constructed after identifying the SoS operational scenarios at a high level of abstraction. Then, the operational activities are mapped to the functions that will fulfill these operations.

The purpose of the mapping process is to lower the level of abstraction and to shift from the customer to the developer domain where constituent systems are identified. Following this process the functional flow diagram is constructed in Fig. 7 Each of these functions is mapped to the constituent system in a separate system view.

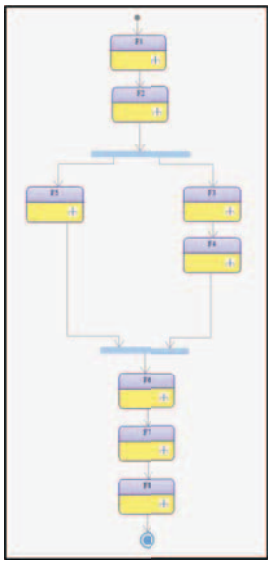

Fig. 7 Functional Flow Diagram

Based on the functional analysis and the mapping between constituent systems and the implemented functions, we create or chose the system architecture pattern that describes the connections between the constituent-system classes, and using the previously mentioned views we build the system reliability block diagram in the UPDM model as shown in Fig. 8. From the RBD the SoS reliability calculations are derived as follows:

$$R_s = R_A + (1 - (1 - R_B)^{x_B}) + (1 - (1 - R_C)^{x_C}) + R_D \quad (5)$$

$R_A, R_B, R_C, R_D$ denote the reliability of the constituent systems A,B,C and D.

We provide the constituent-system catalogue in an Excel sheet, which is generated from the model using the concise plug-in embedded in IBM Rational Rhapsody modelling tool. For each of the constituent-system classes with different instances and different supported redundancy degrees we calculate the reliability using Excel formulas. The resulting values are inserted into the catalogue table. For example, consider constituent-system class B with two options of instances. The first type is with redundancy degrees up to 6 and the second type is with redundancy degrees up to 9 as shown in TABLE I. We use the equation for the reliability calculations in the excel sheet.

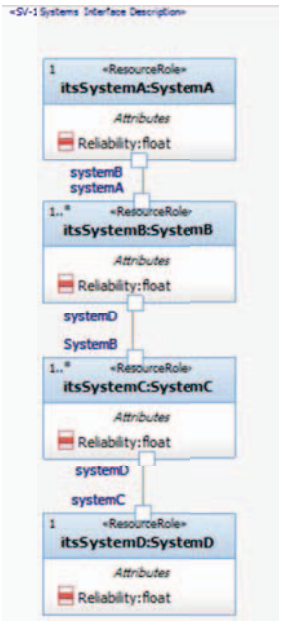


Fig. 8 Reliability Block Diagram

TABLE I EXAMPLE OF CONSTITUENT-SYSTEM CATALOGUE

| int | float | RhpString | int | RhpString | Float |
|---|---|---|---|---|---|
| id | cost | name | systemId | typeName | Reliability |
| 1200000 | 10 | SB11 | 1 | SB11 | -0,020202707 |
| 1200001 | 20 | SB12 | 2 | SB12 | -0,00040008 |
| 1200002 | 30 | SB13 | 3 | SB13 | -8,00003E-06 |
| 1200003 | 40 | SB14 | 4 | SB14 | -1,6E-07 |
| 1200004 | 50 | SB15 | 5 | SB15 | -3,2E-09 |
| 1200005 | 60 | SB16 | 6 | SB16 | -6,4E-11 |
| 1200006 | 8 | SB21 | 7 | SB21 | -0,051293294 |
| 1200007 | 15 | SB22 | 8 | SB22 | -0,00250313 |
| 1200008 | 20 | SB23 | 9 | SB23 | -0,000125008 |
| 1200009 | 25 | SB24 | 10 | SB24 | -6,25002E-06 |
| 1200010 | 30 | SB25 | 11 | SB25 | -3,125E-07 |
| 1200011 | 35 | SB26 | 12 | SB26 | -1,5625E-08 |
| 1200012 | 37 | SB27 | 13 | SB27 | -7,8125E-10 |
| 1200013 | 40 | SB28 | 14 | SB28 | -3,90625E-11 |
| 1200014 | 42 | SB29 | 15 | SB29 | -1,9531E-12 |

## VII. RESULTS

System optimization goals such as maximizing system reliability and minimizing system cost are defined and added to the UPDM model. For our example we added a cost constraint to the system as an upper limit to the system cost. We used the IBM Rational Rhapsody tool for building the UPDM model, MS Excel to provide the constituent-system catalogue and properties and the CPLEX optimizer to solve the optimization problem. A concise plug-in is used to generate the Excel sheet from the UPDM model and to create the optimization problem

for the CPLEX optimizer. After importing the created files in CPLEX, the user can run the optimization problem and the results are exported to text files that can be imported back into the UPDM model using another concise plug-in. This process back-annotates the results as a system architecture model with the specified constituent systems and their interfaces, connections, relationships, and parameters in the Systems Interfaces Description view of UPDM.

To verify our results we implemented the optimization problem also in Matlab with the same parameters. Due to the small size of the example, we could list all possible solutions. As we see in Fig. 9 and Fig. 10 the blue points indicate all possible solutions while the red ones represent only the valid solutions for the given constraints. The green arrow marks the solution chosen by CPLEX which exhibits the minimum cost for the maximum system reliability.

Fig. 9 presents the results with logarithmic values, while Fig. 10 shows the real values.

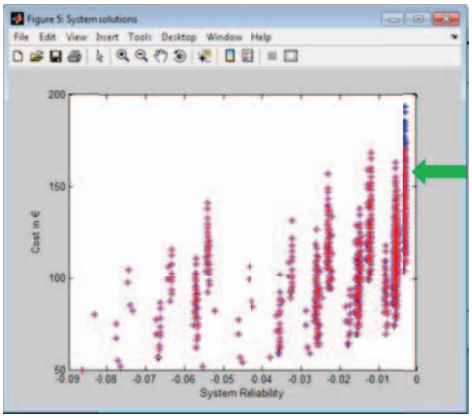


Fig. 9 Matlab results with reliability in logarithmic values

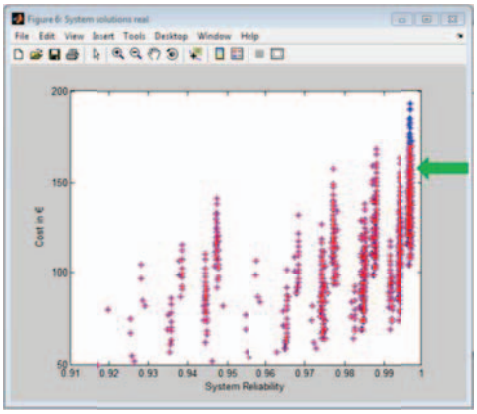


Fig. 10 Matlab results with real reliability values

## VIII. Conclusion

In this paper we presented a model-based system engineering development approach that considers system reliability in the SoS architecture development. UPDM models were extended along with a design process that connects different parts of the model and combines them in a system engineering architecting methodology with support for reliability. The work flow and the models were implemented and an example was evaluated using CPLEX and Matlab.

In future work, we plan to extend the methodology to include additional reliability structures and further extra-functional system properties. The methodology is also flexible for including the Functional Mock-up Interface (FMI) for modelling exchange and co-simulation [23] techniques to extend the model with the ability to perform simulations and model checking for system verification and validation.

## IX. Acknowledgments

This work has been supported in part by the European project DREAMS under the Grant Agreement No. 610640

## X. References:

[1] J. Estefan, “Survey of model-based systems engineering (MBSE) methodologies,” *Incose MBSE Focus Group*, 2007.

[2] G. Booch, J. Rumbaugh, and I. Jacobson, *The unified modeling language user guide*, 1st ed., no. 1. Addison Wesley, 1998.

[3] G. Finance, “SysML Modelling Language explained(2010),” no. October, pp. 1–12, 2010.

[4] M. Maier, “Architecting principles for systems-of-systems,” *Systems Engineering*, vol. 1, no. 4, pp. 267–284, 1998.

[5] DoD Architecture Framework Working Group, “DoD Architecture Framework: Volume I: Definitions and guidelines,” vol. I, no. April 2007, 2003.

[6] British Ministry of Defense, “MOD Architecture Framework,” 2010. [Online]. Available: https://www.gov.uk/mod-architecture-framework#use-and-examples-of-modaf.

[7] NATO, “NATO ARCHITECTURE FRAMEWORK v.3,” 2007.

[8] OMG, “Unified Profile for the Department of Defense Architecture Framework ( DoDAF ) and the Ministry

of Defence Architecture Framework ( MODAF ) V2.0," 2010.

[9] C. Haskins, *Systems engineering handbook*, no. August. 2006.

[10] H. Broodney, D. Dotan, L. Greenberg, and M. Masin, "Generic Approach for Systems Design Optimization in MBSE," 2012.

[11] I. Sanduka and R. Obermaisser, "Model-Based Development of Systems-of-Systems with Real-Time Requirements," *2th IEEE International Conference on Industrial Informatics (INDIN 2014)*, 2014.

[12] F. Tillman, C. Hwang, and W. Kuo, "Optimization Techniques for System Reliability with Redundancy-A Review," *Reliability, IEEE Transactions ...*, vol. R-26, no. 3, pp. 148–155, Aug. 1977.

[13] F. Tillman, C. Hwang, and W. Kuo, "Determining component reliability and redundancy for optimum system reliability," *Reliability, IEEE Transactions ...*, no. 3, pp. 162–165, 1977.

[14] K. B. Misra and M. D. Ljubojevic, "Optimal Reliability Design of a System: A New Look," *IEEE Transactions on Reliability*, vol. R-22, no. 5, pp. 255–258, Dec. 1973.

[15] B. Melhart and S. White, "Issues in defining, analyzing, refining, and specifying system dependability requirements," *Proceedings Seventh IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS 2000)*, pp. 334–340, 2000.

[16] J. Cook, "Multi-state reliability requirements for complex systems," *Proceedings of the 2008 Annual Reliability and ...*, 2008.

[17] W. Kuo and V. R. Prasad, "An annotated overview of system-reliability optimization," *Reliability, IEEE Transactions on*, vol. 49, no. 2, pp. 176–187, Jun. 2000.

[18] A. Birolini, *Reliability Engineering Theory and Practice*. Berlin: Springer, 1999.

[19] H. Kopetz, *Real-time systems: design principles for distributed embedded applications*. 2011.

[20] E. Design, M. Informatization, Q. Guo, X. Li, L. Huang, J. Gao, and X. Xiao, "Research on reliability optimization of weapon system based on heuristic arithmetic," *2011 International Conference on System science, Engineering design and Manufacturing informatization*, pp. 24–26, Oct. 2011.

[21] R. S. Kalawsky, D. Joannou, Y. Tian, and a. Fayoumi, "Using Architecture Patterns to Architect and Analyze Systems of Systems," *Procedia Computer Science*, vol. 16, pp. 283–292, Jan. 2013.

[22] R. Kalawsky and Y. Tian, "Incorporating Architecture Patterns in a SoS Optimization Framework," *Systems, Man, and ...*, 2013.

[23] Modelica Association, "Functional Mock-up Interface." [Online]. Available: https://www.fmi-standard.org/.