SoSAF: A Pharo-Based Framework for Enhancing System of Systems Dependencies Analysis

Mouhamadou F Ball, Patrick Auger, Jannik Laval, Loïc Lagadec

SoSAF: A Pharo-Based Framework for Enhancing System of Systems Dependencies Analysis

Mouhamadou F. Ball[1,*], Patrick Auger[1], Jannik Laval[2] and Loïc Lagadec[1]

[1]ENSTA Bretagne, Lab-STICC, UMR 6285, Brest, France
[2]University Lumiere Lyon 2, DISP Laboratory, Lyon, France

Abstract

System of Systems (SoS) architecture have long been used in the field of system engineering to address increasing and complex requirements. By focusing in SoS with temporal and spatial dynamics of their components, one of their properties is the dependencies between their components. From this perspective, a failure in a component can trigger cascading effects throughout the whole infrastructure. It is crucial to assess how dependencies affect the system's capabilities during the design phase to prevent critical configurations. The first step involves modeling these systems along with their dependencies. Approaches in the literature on SoS dependencies modeling focus on either design, analysis, or execution. Actually, to the best of our knowledge, there are no proposals for a single model that can take all three phases into account. This situation leads to the creation of a gap between these phases, making dependency analysis complex and time-consuming. Furthermore, it is remarkable that there is no model verification process for the approaches studied in the literature. It's worth noticing that, the approaches studied in the literature do not handle real-time interactive simulation into account. Given the dynamic nature of SoS, it's crucial to be able to run the models in an interactive simulation to interact with them and update their parameters. The identified locks regarding this subject are: (i) developing a unified and consistent model for design, execution and dependencies analysis, (ii) handle interaction within the model during simulation.

We introduce in this article the first results of the development of an experimental framework called System of Systems Architecture Framework (SoSAF) for the design, simulation and analysis of the impact of dependencies in SoS operability. To address issue (i), the SoSAF framework introduces a meta-model SoSAF Meta-Model (SM2) developed with the Meta-Object Facility (MOF) specifications, coupled with a model control layer defined with Object Constraint Language (OCL). For interactive simulation (ii) in SoSAF, we present a synchronous and interactive simulation engine on the Pharo environment. By developing this framework on Pharo, we address the issue of interactive simulation. Indeed, thanks to the tools integrated within the Pharo image, such as the inspector, we can interact with a model during the execution of the simulation. To validate our framework's ability to represent and analyze the impact of dependencies, we conducted a case study on a swarm of Unmanned Vehicles (UVs). The case study demonstrates the framework's ability to represent the specifications of SoS and to analyze the impact of dependencies on the operability of all components.

Keywords

System-Of-Systems (SoS), Modeling, Interactive Simulation, Dependencies Analysis,

1. Introduction

Systems have long been used in their monolithic form to provide services and features, but for some time now there has been a trend towards architectures that are operationally and geographically distributed. This transition to increasingly complex systems is the result of growing operational demand. System of systems (SoS) are types of complex systems defined as a set of interconnected systems capable of accomplishing missions that a single system entity cannot [1]. These systems will initially be widely used in the defense industry [2, 3], but later, will experience a rise in civilian applications [4, 5]. However, this increased complexity exposes these systems to new vulnerabilities. The SoS we're interested in here are those with operationally interdependent components evolving in space and time, with a particular focus on Unmanned Vehicles (UVs) fleets. The choice of this type of SoS is motivated by our ongoing

IWST 2024: International Workshop on Smalltalk Technologies, July 9–11, 2024, Lille, France
*Corresponding author.
mouhamadou.ball@ensta-bretagne.org (M. F. Ball); patrick.auger@ensta-bretagne.org (P. Auger); jannik.laval@univ-lyon2.fr (J. Laval); loic.lagadec@ensta-bretagne.org (L. Lagadec)
0009-0005-9245-2892 (M. F. Ball); 0009-0007-8478-9238 (P. Auger); 0000-0002-7155-5762 (J. Laval); 0000-0003-3778-3144 (L. Lagadec)

research concerning the resilience of UVs fleets under high-intensity demands. In the following, we will use the term SoS in a general sense, however, we will refer specifically to SoS that observes the properties and dynamics of UV fleets. Those latter ones are complex systems that are particularly susceptible to cascading failures due to their high dynamics and internal dependencies [6, 7, 8]. Failure of a single component within these architectures can lead to chain reactions and failure of the entire infrastructure.

As exposed by authors in [9], it is essential to consider strategies for enhancing the resilience of these architectures in response to such vulnerabilities. The first step would be to develop tools to analyze these systems and assess their dynamics. In this context, research has been carried out in the field of SoS analysis, in particular on dependencies, to quantify and qualify the effects of such failures.

Our research focuses specifically on the analysis of dependencies, as they represent one of the main sources of complexity in the SoS that is the subject of our study (UVs fleets). By examining the literature, various approaches are proposed for designing models to study dependencies in SoS-type infrastructures [10, 11, 12, 13]. However, it has been observed that these approaches involve models that are specialized in either design, analysis or execution. In fact, there is a gap between the design, execution and analysis models. This means that to move from the design model to an executable version on a simulator, additional model transformation work will be required. This lack of an interoperable formal framework and unified model leads to increased complexity when using these models. We note as well the lack of model verification process on the works we have explored in the literature, which constitutes a limitation for obtaining highly dynamic and consistent models. In fact given the highly dynamic nature of SoS, the model is subject to significant evolution, which is why consistency checking is so important. The simulations reviewed in the review do not take into account interactive simulation, which is crucial for replicating alterations during execution.

The main identified locks surrounding this topic are: (1) developing a unified and consistent model for design, execution and dependency analysis, (2) developing a simulator that can handle interactions with model. Interaction during simulation is crucial to understanding SoS dynamics. The aim is to make simulation scenarios more realistic by changing parameters of executed model. In fact, these parameters variations can be used to simulate alteration occurring either on the SoS components or on the links.

This article presents the first results of the development of an experimental framework for the design, interactive simulation and analysis of the effects of SoS components (CS) dependencies in SoS operability. To address issue (1), the SoSAF framework introduces a meta-model SoSAF Meta-Model (SM2) developed with the Meta-Object Facility (MOF) specification, coupled with a model control layer defined with Object Constraint Language (OCL). For interactive simulation (2) of SoS architectures SoSAF is building a synchronous, interactive simulation engine on the Pharo environment. The choice of implementation in Pharo is motivated by the availability in the Pharo image of powerful tools such as the inspector for interactive simulation, as well as by the simplicity of its syntax, which facilitates the instantiating of our models.

The next Section will discuss the current state of dependencies modeling and analysis in SoS. In Section 3, we will present our SoSAF approach. Then, Section 4 will explore a case study to illustrate how our framework can represent a fleet of unmanned vehicles (UVs) taking into account different levels of granularity, and how to analyze them through an interactive synchronous simulation on SoSAF. Finally, Section 5 will conclude by summarizing the results and addressing perspectives.

2. Related Works and Background

In this Section, we'll look at a few existing approaches to model SoS dependencies, and methods for analyzing the effect of dependencies.

2.1. SoS Modeling and Limitations

The review mainly identified approaches based on the use of graph theory to represent the specific features of SoS-type architectures. These approaches, which are generally parametric, are essentially used to design or study the dynamics and impact of alterations and variations on the overall dynamic of the infrastructure [10, 12, 11]. Some of these approaches use the notion of degree-based graphs to identify important nodes in the network [14], the degree **d** of a node **N** makes it possible to account for the number of links (generated dependencies) originating from **N**.

It is important to note that these various works propose conceptual or analytical (parametric) models. Thus, to integrate these models into a single framework, additional model transformation work will be required. In this way, we can observe the existence of a gap between the conceptual model, the analytical model and the executable model [15, 16]. Moreover, the models presented also lack a verification process to ensure their consistency. This process is important given the dynamic nature of system of systems [17]. It should also be noted that the approaches explored do not deal with interactive simulation, which is a limitation. The possibilities of interaction during the simulation is important, in order to understand the real dynamics of these systems. The variations introduced by the interactions can reflect operational alterations or limitations.

Through the SoSAF framework, our first objective is to establish a meta-model SM2 for the definition of unified models for design, execution and analysis. SM2 will mainly follow the outlines of graph-based approaches, but will also include rules and constraints for the verification of instantiated models. Secondly, we will introduce a simulation engine, designed to run models and perform interactive and synchronous simulations in various scenarios. SoSAF is developed on the Pharo environment, which enables us to take advantage of the tools integrated into the Pharo image, such as inspector. This will facilitate interaction within the running model, and enable it to be reconfigured to perform sensitivity analysis on SoS.

2.2. Models for Dependencies Analysis

In this Section, we introduce 2 parametric models for SoS design and analysis. When designing and analyzing SoS, it is crucial to have models capable of assessing the dependencies between the various components of the SoS. These models are intended to help understand the impact of alterations and their consequences on the overall integrity of the infrastructure. We will examine in more detail two models widely used in the literature for dependency analysis, FDNA and SODA. These are parametric analysis models for studying dependencies. Another model, DDNA, derives from the first 2, adding the concept of execution time. This Section presents these 2 parametric models in detail. Our SM2 meta-model, which will be presented in detail in Section 3.1, will instantiate these models and run them in our synchronous interactive simulator.

2.2.1. Overview Functional Dependencies Network Analysis (FDNA)

This approach, proposed in [10] uses graph-theoretic concepts to model SoS topology. It introduces a parameter called operability (P) for each node in a given mission. The operability is related to performance. The dependencies are quantified using attributes such as Strength Of Dependency (SOD) and Critical Of Dependency (COD). The SOD represents the relationships between the n predecessors and a current node, while the COD accurately highlights the significance of the most crucial dependency. Based on this configuration, equations are established to translate the operability of the nodes, taking these dependencies into account, see figure 1.

2.2.2. Overview System Operational Dependencies Analysis (SODA)

In [12] the authors introduced a new method called SODA, which builds on the fundamental principles of FDNA while addressing some of its limitations, such as the absence of stochastic, and introduces additional parameter called Impact Of Dependency (IOD) as a weight for the edges between related components. The operability of root nodes is equivalent to their self-effectiveness, since they have no dependencies, see figure 2. the self-effectiveness represent the internal status of each component.

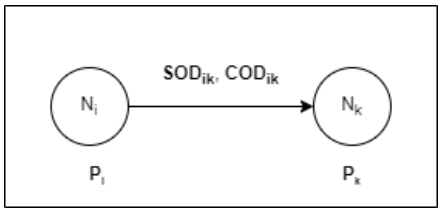


Figure 1: A 2 Components SoS with FDNA.

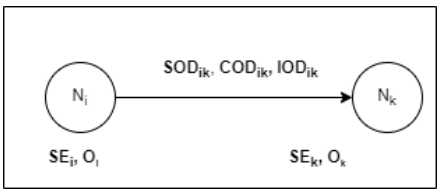


Figure 2: A 2 Components SoS with SODA.

3. SoSAF: System of Systems Architecture Framework

In our context, the SoS typically consists of multiple Components System (CS) that are either aerial, marine, or terrestrial. The components are partially or strongly connected and collaborate to the execution of specific missions. The proposed framework SoSAF offers the possibility of designing, executing and analyzing this type of complex system by integrating features from the dependencies models presented earlier in Section 2.2. Thanks to its unified model, the framework SoSAF facilitates the implementation of multiple dynamic propagation algorithm methods from a single model. The figure 3 below, illustrates the SoSAF ecosystem, showing how the real system model is designed using the SM2 meta-model and results in an interactive simulation using the simulator developed in the Pharo environment. Alterations will be represented as variations in the intrinsic parameters of nodes and links. The use of Pharo as a simulation environment is motivated by the interactivity it offers through its inspector, a tool capable of revealing the data structure behind the elements manipulated during execution. Our model will enable the topology of the infrastructure and its dependencies to be represented with an intuitive syntax. Thanks to the dedicated Pharo environment, the model is run for interactive synchronous simulation, offering the possibility of updating model elements and continuing the simulation. The SM2 meta-model, introduced by SoSAF, is used to create a unified and consistent model that can be run in the Pharo environment (Section 3.1).

3.1. SoSAF Meta-model (SM2)

The domain structure introduced by SoSAF for SoS is based on complex network theory, and thus represents the topology on a directed graph $G = (V, E)$ with the possibility of aggregating nodes in the form of a cluster.

As noted by the authors in [18], most existing meta-models don't specify rules, which can lead to inconsistencies in the model life-cycle. The field of dependency analysis is no exception to this generality. Indeed, the models introduced into the review are not accompanied by verification processes. We have therefore opted to define our meta-model by integrating consistency control rules. The definition of our meta-model is based on the MOF and OCL specifications introduced by the Object Management Group (OMG). Our meta-model presents 2 parts: (1) an object-based definition of concepts and relationships and (2) a set of rules written with the OCL constraint definition language to ensure model consistency. Figure 4 illustrates the structure of a SoS as a composition of several CSs, highlighting the ability of CS to aggregate, as well as the existence of dependencies. The characteristics of those latter may change, reflecting the nature and importance of the link. Each CS has internal attributes such as effectiveness (SE) and operability [11]. System components can be organized into clusters with headers, who is

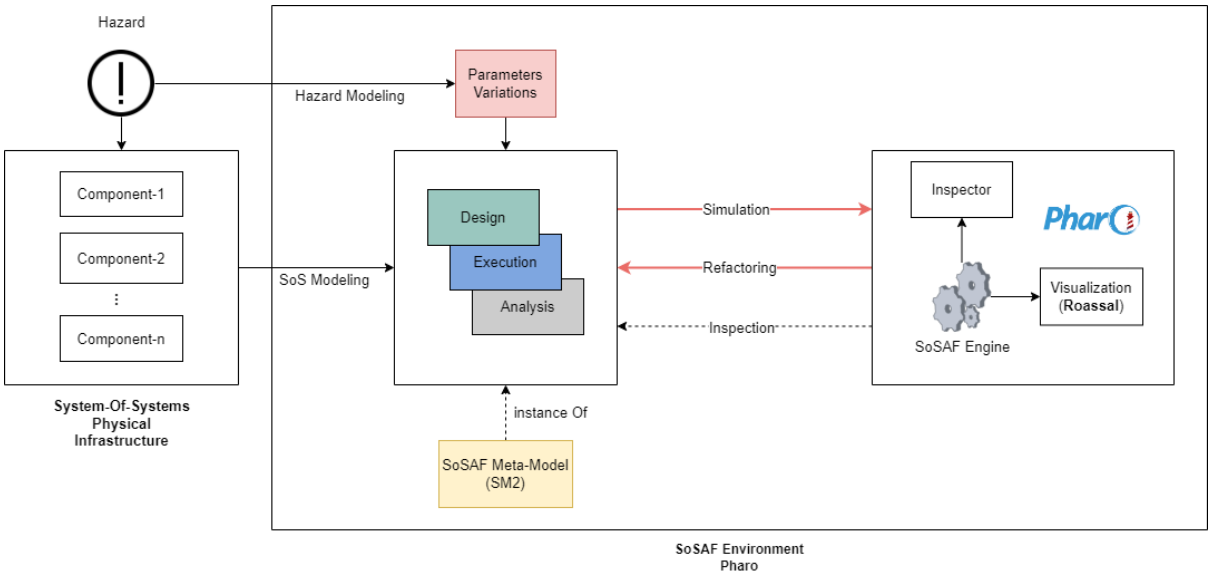


Figure 3: SoSAF Reference Architecture Overview.

elected during the cluster's creation. This concept reflects the different levels of granularity that can be found in systems of systems. In addition, we have added location on a map $(\vec{O},\vec{i},\vec{j})$, to describe geographical independence. The notion of rootEntity, designating a component with no incoming dependencies, is introduced. The incomings attribute of a CS represents the collection of CS on which it depends. The degree-based graph principle introduced by [14] can be handled in our SM2 by the "incomings" parameter, which indicates the number of dependencies and therefore of incident links on a CS.

The defined domain structure will be accompanied by a set of rules expressed in OCL. Through these rules, we will highlight specific properties required by SoS. This is particularly important in the context of interactive simulations, where the model will be frequently reconfigured and executed. It is crucial to perform model verification. The properties discussed in the following rules are a fusion of the specifications of the approaches examined, aggregated in the SM2 meta-model.

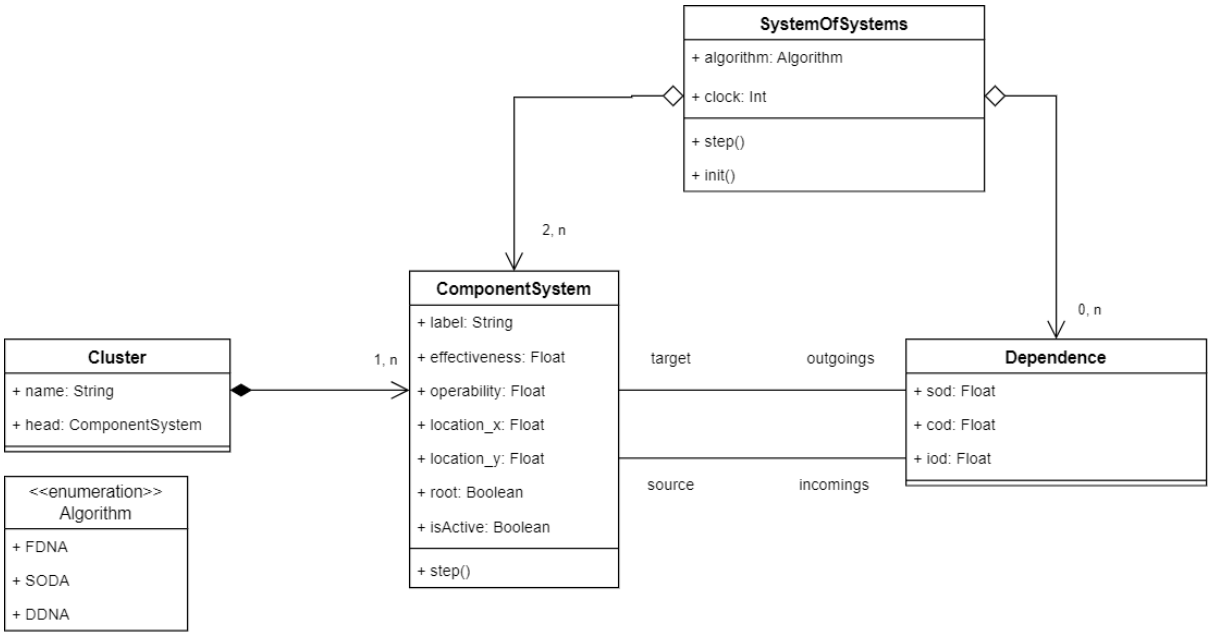


Figure 4: Domain Structure Definition with Meta-Object Facility.

1- Location Constraint

```
context SystemOfSystems inv:
    self.ComponentSystems -> forAll(cs1, cs2 | cs1 <> cs2
        implies
            cs1.x <> cs2.x OR
            cs1.y <> cs2.y )
```

2- Initialization

```
context SystemOfSystems::init inv:
    self.componentSystems -> forAll(cs | cs.root
        implies cs.operability = cs.effectiveness )
    self.componentSystems -> forAll(cs | not cs.root
        implies cs.operability = 0 )
```

3- Stepping

```
context SystemOfSystems::step inv:
    (self.analysisAlgorithm = null OR self.rootEntity->size() = 0)
        implies false
```

```
context SystemOfSystems::step inv:
    self.ComponentSystems -> forAll(cs | not cs.isActive
        implies false)
```

3- Parameters values ranges

```
context ComponentSystem inv:
    (self.operability < 0 OR self.operability > 100)
        implies false
    (self.effectiveness < 0 OR self.effectiveness > 100)
        implies false
```

```
context Dependence inv:
    (self.strength < 0 OR self.strength > 1)
        implies false
    (self.critical < 0 AND self.critical > 100)
        implies false
    (self.impact < 0 AND self.impact > 100)
        implies false
```

3.2. SM2 Instance with Pharo

Our choice to use the Pharo environment as a framework for our model and to support our simulation is based not only on the advantage of interactive simulation, but also on the simplicity of its syntax. As a result, researchers working in the field of systems engineering with limited programming skills will easily be able to use this syntax to define, run and analyze dependencies of system of systems architecture. Model verification are implicitly handled by an exception raising mechanism in Pharo environment. Before execution, the SoSAF engine performs the necessary checks and triggers exceptions specifying the problems if necessary. Below is an example of using the Pharo language to declare a

sample SoS and define properties and dependencies, in accordance with the parametric model previously presented. The value of the model parameters depends on the type of system we want to characterize (could be given by experts in the system under study). To run simulation step after configuring and updating the intrinsic parameters of the SoS, we launch the "step" message. This triggers a simulation step for each CS, enabling the system to evolve towards a state ***T+1*** in accordance with the dynamics described by the analysis algorithm such as FDNA or SODA.

3.2.1. Components System and Dependencies Definition

Creation of an instance of the model with a system of systems of 4 units CS, including the definition of the self-effectiveness attribute, which is a characteristic specific to each control unit. Dependencies are also specified via a collection. The first element designates the CS on which the CS defined by the second element depends, with the remaining elements representing the attributes qualifying the dependency. We can define the parametric model of our choice by adjusting the algorithm attributes: 1 for FDNA approach and 2 for the SODA which adds the impact of dependencies (IOD) parameter on the dependencies.

```
sos := SoSAFModel new.

sos addCS: #(
        #(CS-1 SE) #(CS-2 SE)
        #(CS-3 SE) #(CS-4 SE)
) .

"dependencies definition with analysis algorithm 1 -> FDNA"
sos algorithm: 1.
sos addDependencies: #(
        #(CS-1 CS-2 SOD COD) #(CS-2 CS-3 SOD COD)
); init.

"dependencies definition with analysis algorithm 2 -> SODA"
sos algorithm: 2.
sos addDependencies: #(
    #(CS-1 CS-2 SOD COD IOD) #(CS-2 CS-3 SOD COD IOD)
); init.
```

3.2.2. Clustering

The following code snippet show the creation of two clusters, cluster1 and cluster2, comprising CS-1, CS-2 and CS-3, CS-4 respectively. The cluster head election is deterministic and order-dependent; the first element added to the cluster becomes the header.

```
sos     createCluster: #('cluster1' #(CS-1 CS-2));
        createCluster: #('cluster2' #(CS-3 CS-4)).
```

3.2.3. Update Settings and Run Simulation

It is possible to update internal CSs parameters or dependencies parameters in order to perform a sensitivity analysis.

```
sos
    step;
    update: #(CS-1 new_SE);
    step;
```

```
    updateDependency: #(CS-2 CS-3 new_SOD new_COD);
    step.
```

3.3. Metrics for Dependencies Impact Analysis

The synchronous simulator developed by SoSAF integrates metrics to monitor the evolution of each CS parameter. We have implemented visuals to track model execution and to understand the dynamics of CS and infrastructure evolution. In addition to these internal metrics, we have external observers providing a global view of the state of the system in terms of operational capacity. The latter can be used to observe the topological evolution of a SoS. The following Table 1 lists the parameters currently taken into account in the SoSAF analysis.

Table 1
The Mains indicators to Evaluate SoS in SoSAF.

Indicator	Level	Description
Operability	Component System	The operational capabilities of a CS.
Average Operability	System of Systems	The system's overall operational capabilities at a given time *T*.

4. Case Study: Unmanned Vehicles Swarm

In this Section, we develop a model of a SoS composed of several collaborative UVs. Through SoSAF, we will design and run an interactive simulation to analyze the dynamics of such an infrastructure when nodes or dependencies are perturbed (through parameters values variations).

This example is of particular importance to our studies, as it forms part of our work on the resilience of drone swarms. Through this simple usage scenario, we highlight the various possibilities currently offered by the experimental framework. The UVs system of systems will be made up of 8 CS divided into 2 clusters of 4, with dependencies between the different CSs in the same cluster, as well as between the leading CS (cluster heads). A total of 10 dependencies were defined with random weights. The components of cluster 1 are designated by the letters of the alphabet (A, B, C, D), while those of cluster 2 are designated by the letters (F, G, H, I). The header nodes are represented by A for cluster 1 and F for cluster 2 (leader election based on priority).

4.1. Structure of the UVs Swarm

The code used to define this model is available on the following gitHub repository [1]. Once the 8 CS, clusters and dependencies have been declared, we can display the structure of the SoS with its dependencies as a graph, and inspect the elements using our powerful integrated inspector on Pharo. Nodes with more than two dependencies are marked in red, as they are criticals nodes with high sensibility. Nodes with no dependencies are marked in green, while those with one dependency are marked in gray. The orientation of the arrows indicates the direction of the dependencies. By hovering over the node, its designation is displayed, and thanks to the Pharo inspector, clicking on a node reveals the entire data structure of the component, see figure 5. This inspection capability allows immersion in the model and enables checking the state of our system's components at any point during the simulation.
[1]

4.2. The Sensitivity Analysis

By modifying the intrinsic parameters of the system of systems, it is possible to observe the overall behavior of the system, as well as the reaction of each individual CS. In a real system, we are generally

[1] https://github.com/Cracen26/SoSAF

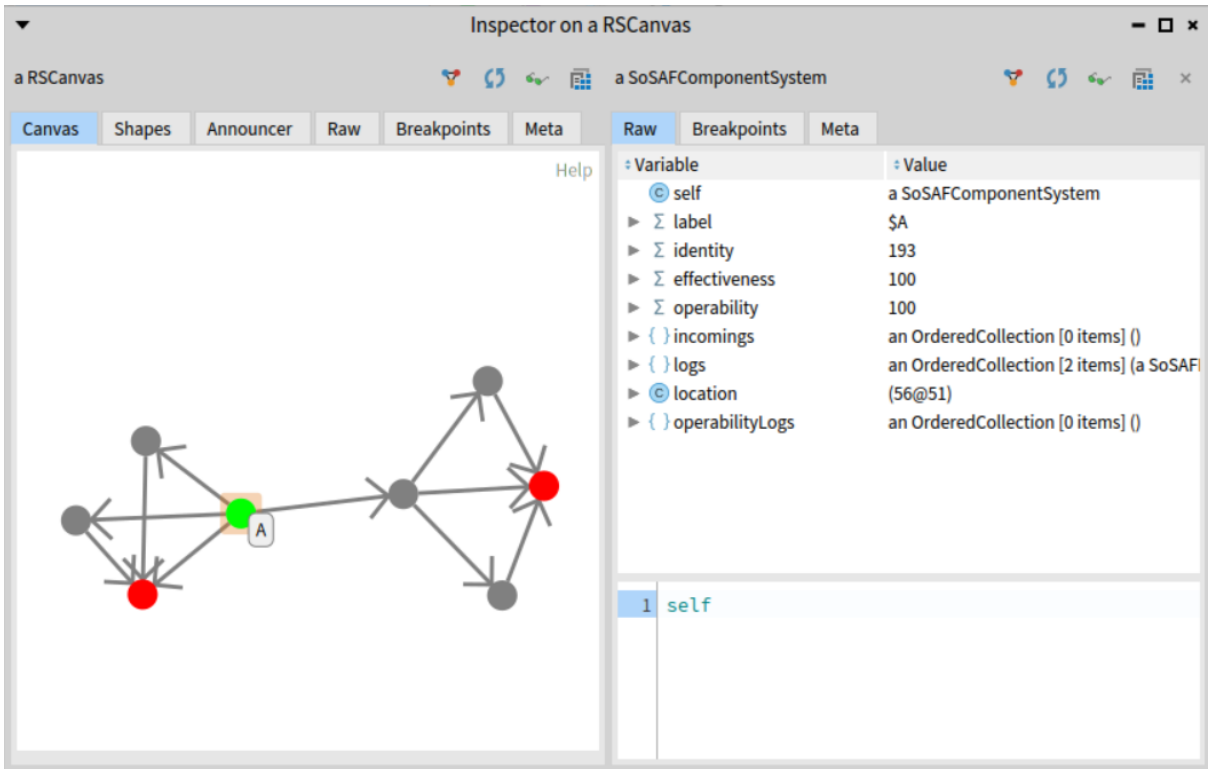


Figure 5: Structure of UVs Swarm in SoSAF with Inspector Focused on CS A (root Component).

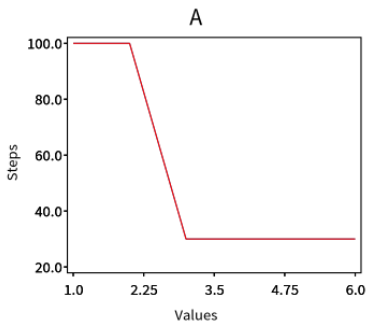
confronted with alterations, whether voluntary (attacks) or involuntary (failures). In all cases, we can abstract these events by varying either the internal efficiency of a node, or the weighting of dependencies to illustrate sensitive communication problems.

The figures below 6a 6b 6c 6d illustrate the evolution of the operability of the CSs in cluster 1 following a decrease in their SE values towards lower values. Initially, the SE values for cluster 1 were as follows: **SE = [A:100, B:60, C:70, D:80]**. We changed them to the following vector: **SE' = [A:30, B:10, C:20, D:40]**. A decrease in internal efficiency below 50% of its maximum value may indicate, for example, a loss of component capacity. The observed evolution is conform to the proposed dynamics of the chosen propagation algorithm (here FDNA). Indeed, any drop in the internal efficiency of a CS is directly reflected in its operability. Variations observed in phases where internal efficiency remains constant are mainly due to incidental dependencies.

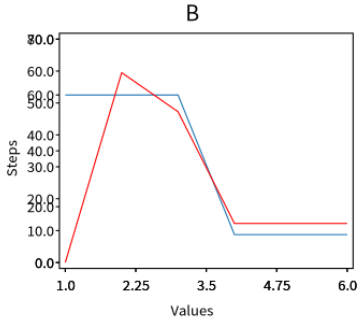
Figures 7a 7b 7c 7d on the other hand, shows how the CS of cluster 2 evolves following adjustments to the dependencies. These adjustments aim to make the dependencies more binding, which means that, when using the FDNA algorithm, SOD tends towards 1 and COD tends towards 0. The dependencies in cluster 2 were initially: (F, G, 0.2, 50), (F, H, 0.1, 60), (G, I, 0.3, 40), (H, I, 0.1, 70), (F, I, 0.2, 90). The new distribution of values is as follows: (F, G, 0.7, 10), (F, H, 0.6, 30), (G, I, 0.8, 50), (H, I, 0.9, 50), (F, I, 0.5, 20). Referring to the dependency adjustments, one might anticipate a drop in operability. However, we observe the opposite behavior, which can be explained in the context of the FDNA algorithm by the very high internal efficiency of the CSs making up the nodes of cluster 2. (**SE' = [F:100, G:80, H:70, I:50]**

Figure 8 displays the operability status of each CS at a specific point in the simulation, providing an overview of the SoS performance and facilitating the identification of components in critical states.

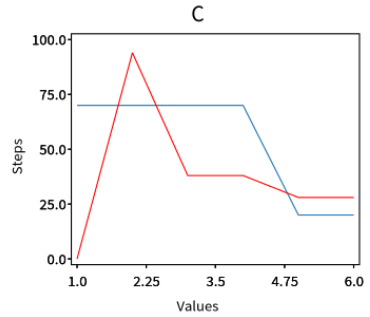
This case study demonstrates our framework's ability to describe highly dynamic SoS such as UVs fleets, and to run a synchronous interactive simulation for dependencies analysis, all from a single model instantiated according to the SM2 meta-model and whose consistency is managed by the verification layer.



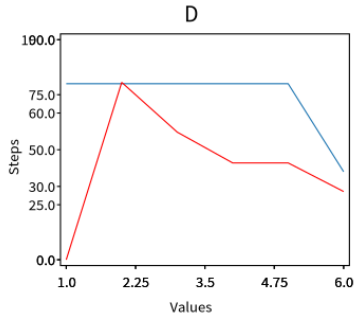
(a) OP and SE of CS A



(b) OP and SE of CS B



(c) OP and SE of CS C



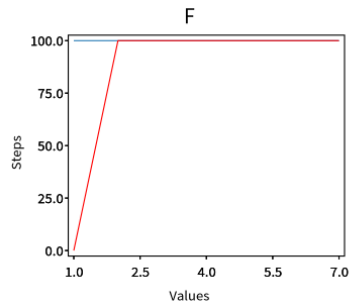
(d) OP and SE of CS D

Figure 6: Operabilities Evolution of CS in Cluster 1 Following a Reduction in Their Self-Effectiveness (the red curve represents Operability and the blue curve the SE of the CS).

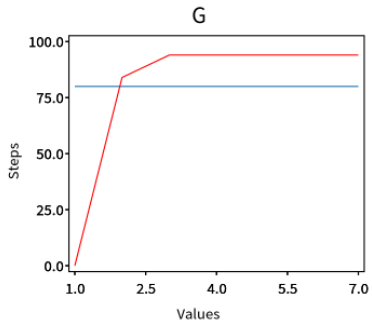
5. Conclusion and Perspectives

This paper discusses the creation of a framework for dependencies analysis in system of systems configuration. The framework incorporates a meta-model designed to instantiate a comprehensive "all-in-one" model capable of performing design, analysis, and execution tasks. This meta-model is associated with a verification layer to address consistency issues, taking into account the high dynamics of SoS. We are also proposing an interactive simulator in the Pharo environment, enabling us to synchronously simulate the architectures under study and understand their dynamics in the face of dependencies/components failures and their impact on mission. All these achievements are integrated into the SoSAF framework. The case study of UVs Swarm presented here demonstrates the ability of our approach to define problems linked to dependencies between system of systems, to run an interactive synchronous simulation and to analyze the evolution dynamics according to the chosen parametric propagation model.

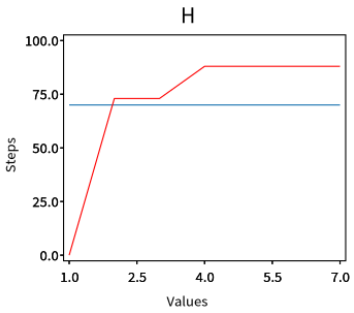
It is important to note that, in the current SoSAF method, SoS data and specifics are added manually in the model. The long-term objective is to enable dynamic updating of data intrinsic to the model, which would be automatically integrated into the model. The Pulse approach, developed by a team from the Decision and Information Systems for Production systems laboratory (DISP) [19], could provide the beginnings of a solution. It proposes a meta-model for aggregating data collected in systems, with an emphasis on interoperability. By exploiting this approach, it would be possible to move towards the creation of digital twins of SoS. The view of a SoS component as a unique element could be re-evaluated to allow for internal intervention in terms of resilience. In fact, this precision in the abstraction of components will facilitate consideration of the reuse of a failing component for other purposes.



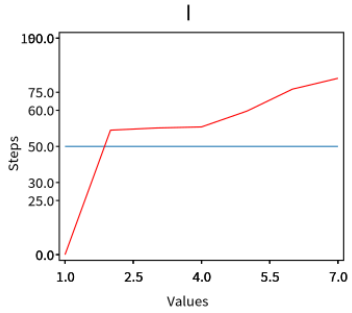
(a) OP and SE of CS F



(b) OP and SE of CS G



(c) OP and SE of CS H



(d) OP and SE of CS I

Figure 7: Operabilities Evolution of CS in Cluster 2 Following Dependencies Parameters Updates (the red curve represents Operability and the blue curve the SE of the CS).

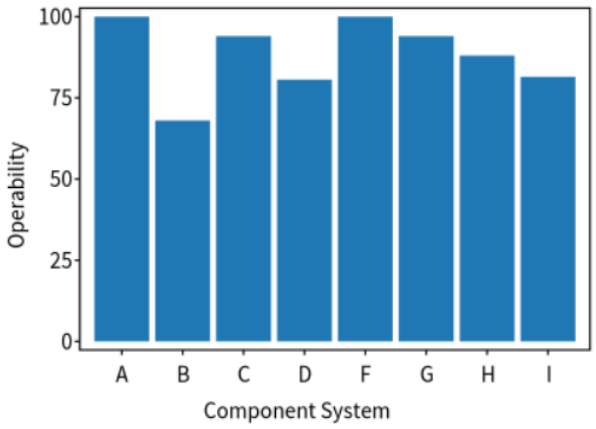


Figure 8: Operational Statuses at a Specific Time-Step Indicating Overall Performance of SoS Components.

References

[1] M. W. Maier, Architecting principles for systems-of-systems, Systems Engineering: The Journal of the International Council on Systems Engineering 1 (1998) 267–284.

[2] J. M. Pullen, O. M. Mevassvik, Coalition command and control—simulation interoperation as a system of systems, in: 2016 11th System of Systems Engineering Conference (SoSE), IEEE, 2016, pp. 1–6.

[3] D. G. Lubas, Department of defense system of systems reliability challenges, in: 2017 Annual Reliability and Maintainability Symposium (RAMS), IEEE, 2017, pp. 1–6.

[4] P. Salvaneschi, Modeling of information systems as systems of systems through dsm, in: Proceed-

ings of the 4th International Workshop on Software Engineering for Systems-of-Systems, 2016, pp. 8–11.
[5] F. Alkhabbas, R. Spalazzese, P. Davidsson, Emergent configurations in the internet of things as system of systems, in: 2017 IEEE/ACM Joint 5th International Workshop on Software Engineering for Systems-of-Systems and 11th Workshop on Distributed Software Development, Software Ecosystems and Systems-of-Systems (JSOS), IEEE, 2017, pp. 70–71.
[6] L. Duenas-Osorio, S. M. Vemuru, Cascading failures in complex infrastructure systems, Structural safety 31 (2009) 157–167.
[7] L. Dueñas-Osorio, J. I. Craig, B. J. Goodno, Seismic response of critical interdependent networks, Earthquake engineering & structural dynamics 36 (2007) 285–306.
[8] L. Xing, Cascading failures in internet of things: review and perspectives on reliability and resilience, IEEE Internet of Things Journal 8 (2020) 44–64.
[9] T. Xu, Y. Chen, C. Lu, H. Li, J. Lv, Importance measure of equipment task based on operational dependency of sos, IEEE Access 9 (2021) 15452–15466.
[10] P. R. Garvey, C. A. Pinto, Introduction to functional dependency network analysis, in: The MITRE Corporation and Old Dominion, Second International Symposium on Engineering Systems, MIT, Cambridge, Massachusetts, volume 5, 2009.
[11] C. Guariniello, D. DeLaurentis, Dependency analysis of system-of-systems operational and development networks, Procedia Computer Science 16 (2013) 265–274.
[12] C. Guariniello, D. DeLaurentis, Supporting design via the system operational dependency analysis methodology, Research in Engineering Design 28 (2017) 53–69.
[13] Y. Wang, W. X. Zhang, Q. Li, Functional dependency network analysis of security of navigation satellite system, Applied Mechanics and Materials 522 (2014) 1192–1196.
[14] R. J. La, Cascading failures in interdependent systems: Impact of degree variability and dependence, IEEE Transactions on Network Science and Engineering 5 (2017) 127–140.
[15] Z. Fang, System-of-systems architecture selection: A survey of issues, methods, and opportunities, IEEE Systems Journal 16 (2021) 4768–4779.
[16] L. W. Wagenhals, A. H. Levis, Service oriented architectures, the dod architecture framework 1.5, and executable architectures, Systems Engineering 12 (2009) 312–343.
[17] B. H. Thacker, S. W. Doebling, F. M. Hemez, M. C. Anderson, J. E. Pepin, E. A. Rodriguez, Concepts of model verification and validation (2004).
[18] J. Cadavid, B. Combemale, B. Baudry, Ten years of Meta-Object Facility: an analysis of metamodeling practices, Ph.D. thesis, INRIA, 2012.
[19] J. Laval, N. Amokrane, B. Thiam Niang, M. Derras, N. Moalla, Data interoperability assessment, case of messaging-based data exchanges, Journal of Software: Evolution and Process 35 (2023) e2538.