IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received November 6, 2020, accepted November 20, 2020, date of publication December 7, 2020,
date of current version December 21, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3043036

Early Quality Evaluation of System of Systems Architecture Within Trade Study Process

JOVITA BANKAUSKAITE[1,3], AURELIJUS MORKEVICIUS[2,3], AND RIMANTAS BUTLERIS[1]
[1]Centre of Information Systems Design Technologies, Kaunas University of Technology, LT-51423 Kaunas, Lithuania
[2]Department of Information Systems, Kaunas University of Technology, LT-51368 Kaunas, Lithuania
[3]Dassault Systemes, LT-49425 Kaunas, Lithuania

Corresponding author: Jovita Bankauskaite (jovita.bankauskaite@ktu.edu)

ABSTRACT In this age of innovation, the application of model-based system engineering is key for solving complex problems. Complexity issues are particularly prevalent when there are several systems that must communicate independently. From a system engineering point of view, this is recognized as a system of systems level. At this level, the major concern is the architecture assessment that is usually performed by trade study analysis. However, a trade study is a time-consuming and expensive activity. In consideration of these obstacles, we propose a new trade study process (Bankauskaite and Morkevicius, 2020) for the system of systems architecture developed in Unified Architecture Framework models. This process is designed to support automated methods that allow for greater transparency, better uncertainty analysis, and increased flexibility in adapting to scope changes. One of the crucial stages of the trade study process is the initial assessment of alternative architectures, called the Base Check. The purpose of this stage is to narrow the set of solutions to select only plausible alternatives that are of sufficient quality and can be further evaluated. This paper presents the key rules applicable to initial architecture assessment, together with automated methods. Provided rules are verified by running an experiment on real-world models to confirm their eligibility.

INDEX TERMS Trade study, alternative analysis, unified architecture framework (UAF), architecture evaluation, enterprise architecture.

I. INTRODUCTION

Architecture evaluation is one of the key stages in the architecture development lifecycle. Its main goal is to identify the best candidate architecture according to the needs and underlying assumptions. To accomplish that, it is necessary to conduct an analysis of trade study by comparing the identified alternatives of architecture and highlighting their key benefits and deficiencies [2].

Trade study analysis is a time-consuming and expensive activity, requiring a lot of resources and knowledge. Typically, architectures are individually evaluated on each parameter of interest, with the results compared [3]. A recent US Air Force trade study revealed that the average cost was $15M and it took 21 months to complete [4]. In order to automate this complex process, it is necessary to develop a digital, unambiguous, and precise model-based environment consisting of an architecture framework (AF), modeling language, and tool. This allows for faster assessment of more possible architectures and may lead to the identification of the most promising solution architectures that may not be identified using traditional methods [3].

Nowadays, there are several AFs and languages that are extensively used when it comes to the system of systems (SoS), such as the NATO Architecture Framework (NAF) or the Department of Defense Architecture Framework (DoDAF) [5]. Over time, system of systems engineering (SoSE) continues to evolve to simplify intricate processes and concepts. Three years ago, a new AF, called the Unified Architecture Framework (UAF), emerged and became the official Object Management Group (OMG) standard. UAF has been designed to support trade studies at various levels, including three core domains: Strategic, Operational, and Resources [6]. The mechanism to capture the required data is provided; however, the process of doing that is missing [1]. Based on advances in model-based systems engineering (MBSE), best practices of existing trade study processes, and UAF principles, the new trade study process for the SoS architecture is introduced in [1].

The associate editor coordinating the review of this manuscript and approving it for publication was Bing Li.

The proposed process is compatible with UAF modeling language and framework. One of the key stages of the process is the base check. The goal of this stage is to determine the quality of alternative architecture and narrow the set of solutions to select only plausible alternatives, which are of sufficient quality and can be further evaluated. The base check is important because the poor quality of architecture can lead to misleading conclusions in further analysis.

In this paper, we are focusing on the SoS architecture evaluation. The purpose of our research is to propose a new set of UAF-based validation rules that allows determining the quality of architecture at an early stage of the trade study process and narrow the set of alternatives that will be evaluated in the next stage of analysis. A smaller number of alternatives will require a shorter trade study analysis.

The structure of this paper is as follows: in Section 2, the related works are analyzed; in Section 3, the proposed set of validation rules is presented; in Section 4, experimental evaluation and application of the proposed validation rules are described; in Section 5, the achieved results, conclusions, and future work directions are indicated.

II. LITERATURE REVIEW

This section consists of three parts. The first part provides an overview of the UAF, which is cross-industries oriented standardized AF. The second part revises architecture evaluation methods, while the third part presents an overview of automated techniques for evaluating architecture in the MBSE environment.

A. UNIFIED ARCHITECTURE FRAMEWORK

Taking industry demand into account and addressing the changing landscape of existing domain specific AFs, in September of 2013, a Request for Proposal for Unified profile for DoDAF and MODAF (UPDM 3.0) (later renamed to Unified Architecture Framework (UAF)) was created. Seven years later, a new UAF 1.0 has become an official Object Management Group (OMG) standard and is about to become ISO/IEC 19517 standard.

Applying the Unified Architecture Framework, using an MBSE approach moves the architecture modeling effort to one that is an integral part of SE [7]. This helps the systems integrator develop interoperable systems with traceability to requirements and across views, using one integrated architecture model that enables impact, gap, engineering analysis, trade studies, and simulations [8]. Moreover, the scope of UAF is expanded beyond defense architectures. It is generalized to be applicable to architecting system of systems of any domain. UAF version 1.0 is industry domain agnostic [9]. The current version of the standard is 1.1.

UAF consists of three main components

- framework – a collection of domains, model kinds, and view specifications,
- metamodel – a collection of types and individuals used to construct views according to the specific view specifications,
- profile – SysML based implementation of the metamodel to apply model-based systems engineering principles and best practices while building the views.

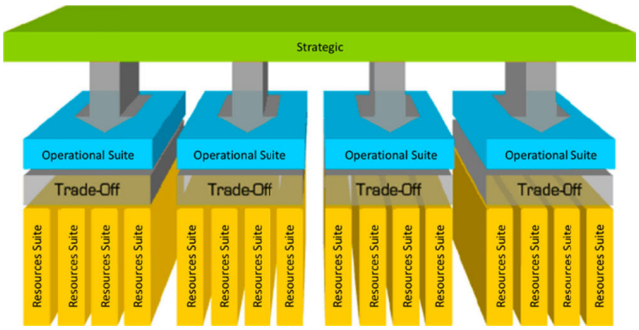

FIGURE 1. Architecture frameworks abstraction layers in the context of Trade Study process [1].

The Grid format is the most suitable way to describe the framework component of the UAF. It is organized into rows and columns, where rows are Domains and columns are Model Kinds. The intersection of a row and column is called a view specification. A UAF grid summarizes all view specifications available in existing AFs. It serves as a foundation for building domain-specific frameworks by selecting only the viewpoints required for a specific context.

Core domains, such as strategic, operational, and resources, represent different layers of abstraction. While architecture development progresses from strategic to resources domains, t multiple trade studies can be performed (Figure 1) including trade studies between different operational scenarios or between different resource configurations. UAF standard does not define how trade studies are carried out. It does, however, provide a foundation to capture data necessary for trade studies.

B. ARCHITECTURE EVALUATION METHODS

An architecture evaluation is a structured process by which one or more architectures are assessed, taking into account the specified evaluation objectives to ensure that the design is in line with the desired quality [10], [11]. One of the key motives for evaluating architecture is that poorly designed or imprecisely assembled architectures will cause the system to malfunction. Fixing defects at an early phase of the system development cycle is less painful and expensive than to solve problems that occur in later phases [12], [13]. The evaluation of architecture uses a variety of methods, which can be divided into three main categories: scenario-based, simulation-based, and mathematical modeling based.

1) SCENARIO-BASED ARCHITECTURE EVALUATION

Scenario-based architecture evaluation is a method based on the concept of a scenario. Scenarios are related to architectural concerns, such as quality attributes. The aim of the scenarios is to highlight the consequences of the architectural decisions related to the design [14]. Each identified scenario is checked to determine whether it is supported by a system architecture or not [15]. A number of scenario-based evaluation methods have been developed. The most well-known include the Architecture Trade-off Analysis Method (ATAM), Scenario-based Software Architecture

Analysis Method (SAAM), Architecture Level Modifiability Analysis (ALMA), and others [16], [17].

2) SIMULATION-BASED ARCHITECTURE EVALUATION

Simulation-based architecture evaluation integrates architecture evaluation techniques into a simulation. Therefore, the user may explore the designs without actually physically building the systems [18]. The execution of a model not only helps to better understand the architecture, it is also very useful for conducting various analyses. Architecture models allow performing analysis to determine the architecture correctness or completeness, to check the appropriateness of resources, to evaluate performance [19], to identify system overload or to determine bottlenecks in the processes. These benefits have led to widespread use of simulation-based evaluation in various case studies [20]–[24].

3) MATHEMATICAL MODEL-BASED ARCHITECTURE EVALUATION

Mathematical model-based architecture evaluation uses mathematical methods to quantitatively assess the quality attributes of an architecture [25]. Quantitative assessment is particularly important in critical systems such as medical, aircraft, and space mission [17]. Therefore, several architecture evaluation methods based on mathematical models have been developed [19], [26], [27]. Based on these methods the architecture includes mathematical equations.

Depending on which evaluation method is used and how the evaluation is carried out, the results can be divided into quantitative and qualitative. Qualitative results provide an answer to the question "Is architecture X better than architecture Y?" Quantitative results answer "How much better is architecture X than architecture Y?" The use of methods that provide quantitative results allows users to compare architectures and track qualitative attributes.

In this paper, we decided to evaluate the qualitative attributes of architecture, achieved by using a scenario-based approach together with simulation.

C. AUTOMATED TECHNIQUES TO EVALUATE AN ARCHITECTURE IN MBSE ENVIRONMENT

Automated architecture evaluation reduces the manual effort to evaluate and compare different architectures. The combination of architecture designed in the MBSE environment with existing automated architecture evaluation techniques allows users to speed up the evaluation phase. Table 1 provides an overview of well-known automated techniques [28]–[33] used to evaluate model-based architecture.

In this study, we decided to evaluate the proposed architecture evaluation rules by using validation-based metrics techniques. This approach allows combining the summarized results with traceability to failed elements.

III. BASE CHECK OF ENTERPRISE ARCHITECTURE MODELS

This section consists of two parts. The first part briefly introduces the UAF-based trade study process and its main activities. The second part provides a set of rules for architecture evaluation based on the UAF-based trade study process.

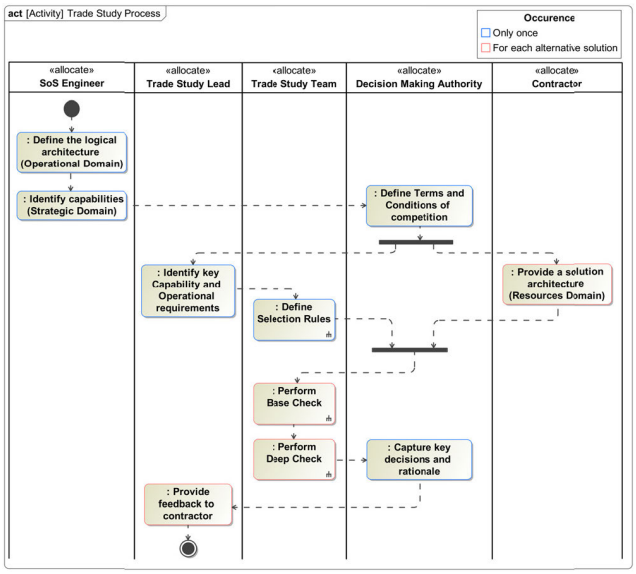

FIGURE 2. Trade Study Process [1].

A. UAF-BASED TRADE STUDY PROCESS

A trade study is an evaluation of alternatives based on criteria and systematic analysis to select the best alternative for attaining determined objectives. Potential solutions of a trade study are judged by their overall satisfaction with a series of desirable characteristics [34]–[36].

In the article [1], an analysis of trade study processes [2], [37] disclosed that most of them are universal and not specific to any domain. Additionally, their application in the model-based environment can be difficult and at times may require additional process modifications. In support of these issues, a new trade study process is introduced [1] that is strictly based on the principles of UAF. It defines each stage of the trade study process (Figure 2) and identifies five key roles involved in the process: SoS Engineer, Trade Study Lead, Trade Study Team, Decision Making Authority, Contractor. In addition, the required inputs and output are provided for each stage of the trade study. The process guides users in performing a trade study when it is decided to run a competition in the acquisition.

The first activity is to define a logical architecture (Strategic and Operational Domain models) that outlines what needs to be achieved by the solution. The second activity is to define the rules on how the competition is organized, who can participate, etc. The most important stage is to determine the rules under which the most appropriate solution is chosen.

Improperly set rules and criteria can distort the results and highlight a solution that does not meet the key requirements and characteristics. Meanwhile, contractors who want to participate in a competition should provide a solution architecture (Resources Domain model). Each solution should be modeled in accordance with the requirements (Strategic and Operational Domain models). Once the contractors submit their solutions according to the requirements the investigation phase begins.

TABLE 1. Automated techniques to evaluate model-based architecture.

	Validation Rule	Metrics	Metrics based on validation rules	Automatic requirement verification
Description	This is the most common way to check the model against rules or constraints. A validation rule can contain a formula or expression that evaluates a model and returns the value "True" or "False". Rules can be divided into different groups according to their criticality: Error, Warning, Info, etc. Validation rules also provide a message when an error occurs.	Metrics are measures of quantitative assessment that are used to evaluate the current state of the model. A metric provides information about a specific aspect of the model. When calculated at regular intervals, they can help track the evolution of the development of the model.	It is a combination of validation rules and metrics. Metrics use an expression of the validation rule for the calculations. Validation rules and metrics are connected, so the results displayed are relevant to the same context.	Usually, system models are related to text requirements. Some of the requirements can be measured with parameters. Parameters set out in the model can be calculated to verify the system requirements and decide whether they are satisfied or not.
Usage	Model evaluation (functional, structure, requirements)	Model KPI, Model status tracking, Model management	Model evaluation, Model KPI, Model status tracking, Model management	Verification of parameterized requirements
Advantages	Flexibility - the application of rules is quite extensive. Rules can be applied by checking a model from different domains. Clarity - when an error is found, it is bounded to the concrete element of a model. A customized error message can be displayed. Integration with tool – results of validation rules can be used with other operations, such as prohibiting commits of changes to the server when certain errors occur.	Structured results – data is represented in numeric values, easy to read. Data analysis – metrics can be exported to another file format to carry out an analysis. It is easy to use various tools to highlight certain metrics and track the status of the model.	Flexibility - the rules can be applied to various models from different domains. Clarity - when an error occurs, an error message is displayed. Data analysis – metrics can be exported to another file format to perform various data analysis.	User-friendly – this is an easy way to verify the parameterized requirements. No programming knowledge is required. Model logic check – using this method, the logic of a model is verified, not the correctness or completeness. Early requirements verification – it is possible to check a model in the early phases and ensure that certain requirements are met.
Disadvantages	Complicated – to create a rule, you need to write certain expressions or formulas, which can be quite complex for users who do not have basic programming knowledge.	Complicated - this requires at least basic programming skills to create custom metrics. Lack of Clarity - it is not clear which element failed because only the numerical data is displayed.	Complicated – to create a rule, you need to write certain expressions or formulas, which can be quite complex for users who do not have basic programming skills.	Small subset of elements – usually applied between the requirement and an element with a numeric value, as value property. Only used for quantitative assessments – this method only checks the parameterized requirements for verifying that the total mass, speed or other parameters comply with the requirements.

This stage consists of two main parts: Base Check and Deep Check. The Base Check stage is intended to evaluate the quality of alternatives and narrow the set of submitted solutions. The Deep Check stage is intended to evaluate the selected alternatives according to established evaluation criteria. Finally, based on the results of the study, the decision-making authority selects the most appropriate alternative that meets the essential requirements.

B. VALIDATION RULES FOR THE BASE CHECK OF SOS ARCHITECTURE MODELS

This section consists of two main parts (Figure 3). The first part provides a set of rules for the vertical architecture evaluation. In this part, the validation rules check the existence of the necessary traceability links between several domains therefore they are called vertical rules. The second part provides the set of rules for the horizontal architecture evaluation. Rules that check the existence of the necessary traceability links between elements from the same domain are called horizontal.

All predefined rules have an assigned level of severity according to their importance and the impact of the rule violation. We use three different severity levels: High, Medium, and Low. Based on the violations of the executed rules defined in sections 3.2.1.1, 3.2.1.2 and 3.2.2, the quality level of a particular alternative solution can be calculated. The ratio between violated and executed rules is a common comparison method for comparing two quantities [38].

The quality index of SoS architecture can be evaluated using the following equation (1):

$$Q = 1 - \frac{\sum_{i=1}^{3} s_i \times vr_i}{\sum_{i=1}^{3} s_i \times er_i} \quad (1)$$

where:
Q – Quality index for a single alternative solution;
i – Severity level;
s – Weight of severity level;
vr_i – Number of violated validation rules by severity level i;
er_i – Number of executed validation rules by severity level i;

TABLE 2. 'Operational vs Resources' validation rules.

No	Validation Rule	Severity (Weight)	Description
1.	∀x[OperationalPerformer(x) ∧ Atomic(x) → ∃y[ResourcePerformer(y) ∧ Implements(y,x)]] – *OperationalPerformer(x)* – x is an element with stereotype <<OperationalPerformer>>; – *Atomic (x)* – x is atomic and it cannot be divided; – *ResourcePerformer(y)* - y is an element with stereotype <<ResourcePerformer>>; – *Implements(y,x)* – y element implements x by Implements relationship;	**High** (weight=3)	**Rule Description**: Each atomic *Operational Performer* must be implemented by *Resource Performer*. **Constrained Element**: *Operational Performer* **Weight Justification**: It is important that the logical entity identified in the upper layer of the abstraction is connected to resources. The linkage ensures that logical entity is implemented by a semantically equivalent element at the lower level of abstraction. **Type**: Direct **Technical Description**: Atomic *Operational Performer* element should be linked with a *Resource Performer* element using the <<Implements>> relationship.
2.	∀x[OperationalActivity(x) ∧ Atomic(x) → ∃y[Function(y) ∧ Implements(y,x)]] – *OperationalActivity(x) – x is an element with stereotype <<OperationalActivity>>;* – *Atomic (x) – x is atomic and it cannot be divided;* – *Function(y) - y is an element with stereotype <<Function>>;* – *Implements(y,x) – y element implements x by Implements relationship;*	**High** (weight=3)	**Rule Description**: Each atomic *Operational Activity* must be implemented by *Function*. **Constrained Element**: *Operational Activity* **Weight Justification**: It is important that all identified *Operational Activities* in the upper layer of abstraction are associated with a *Function*. The linkage ensures that *Operational Activity* is implemented by a semantically equivalent element at the lower level of abstraction. **Type**: Direct **Technical Description**: Atomic *Operational Activity* element should be linked with a *Function* element using the <<Implements>> relationship.

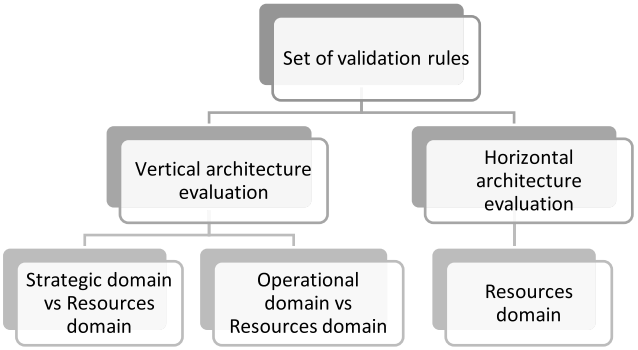

FIGURE 3. Structure of the set of validation rules for architecture evaluation.

Depending on the severity of the rule, its weight varies when calculating the quality index of the architecture. The weight of the rule is a freely selectable measure.

1) VERTICAL ARCHITECTURE EVALUATION

Vertical architecture evaluation is used to determine whether the contractor has established essential traceability links between their proposed solution and the provided requirements. We divided the proposed traceability rules for vertical architecture evaluation into two categories by their inspected domains: Strategic Domain vs Resources Domain, Operational Domain vs Resources Domain. Validations rules for both categories are provided below.

1.1) OPERATIONAL DOMAIN VS RESOURCES DOMAIN

The operational domain of UAF concerns the logical architecture of the enterprise. It describes requirements, operational behavior, structure, and the exchanges required to support capabilities. The operational domain defines all operational elements in an implementation-independent manner [39].

The established 'Operational vs Resources' rule set (Table 2) aims to verify the existence of the necessary traceability links between the logical architecture (what needs to be accomplished) and the solution architecture (resources needed to implement the operational requirements). The set of rules contains ten traceability rules based on the UAF metamodel: 1) direct (direct traceability relationship between two elements, rules developed from the Operational domain point of view), 2) direct reverse (direct traceability relationship between two elements, rules developed from the Resource domain point of view), and 3) derivatives (derived traceability relationships with regard to other relationships).

Input: Operational domain model; Resources domain model; Predefined 'Operational vs Resources' validation rules.

Output: Official report on the results of the 'Operational vs Resources' architecture quality evaluation.

1.2) STRATEGIC DOMAIN VS RESOURCES DOMAIN

The strategic domain of UAF concerns the capability portfolio management process. This is where the enterprise objectives and vision are captured and the capabilities needed to achieve these objectives are defined [39]. The strategic domain shows the relationships between capabilities itself and between the capabilities and the resources required to realize them.

The established 'Strategic vs Resources' rule set (Table 3) aims to verify the existence of the necessary traceability links between capabilities and resources. The set of rules contains two traceability rules based on the UAF metamodel: 1) direct (direct traceability relationship between two elements, rules developed from the Strategic domain point of view); and 2) direct reverse (direct traceability relationship between two

TABLE 2. ***(Continued.)*** **'Operational vs Resources' validation rules.**

3.	∀x[InformationElement(x) → ∃y[DataElement(y) ∧ Implements(y,x)]] – *InformationElement (x) – x is an element with stereotype << InformationElement >>;* – *DataElement (y) - y is an element with stereotype << DataElement >>;* – *Implements(y,x) – y element implements x by Implements relationship;*	**Medium** (weight=2)	**Rule Description**: *Information Element* must be implemented by *Data Element*. **Constrained Element**: Information Element **Weight Justification**: It is important that all *Information Element* in the upper layer of abstraction is linked with the *Data Element*. The linkage ensures that *Information Element* is implemented by a semantically equivalent element at the lower level of abstraction. **Type**: Direct **Technical Description**: *Information Element* should be linked with a *Data Element* using the <<Implements>> relationship.
4.	∀x[OperationalExchange(x) → ∃y[ResourceExchange(y) ∧ Implements(y,x)]] – *OperationalExchange (x) – x is an element with stereotype <<OperationalExchange >>;* – *ResourceExchange (y) - y is an element with stereotype <<ResourceExchange >>;* – *Implements(y,x) – y element implements x by Implements relationship;*	**Medium** (weight=2)	**Rule Description**: *Operational Exchange* must be implemented by *Resource Exchange*. **Constrained Element**: Operational Exchange **Weight Justification**: It is important that the *Operational Exchange* in the upper layer of abstraction is linked with the *Resource Exchange*. The linkage ensures that the *Operational Exchange* is implemented by equivalent element at the lower level of abstraction. **Type**: Direct **Technical Description**: *Operational Exchange* should be linked with a *Resource Exchange* using the <<Implements>> relationship.
5.	∀x[ResourcePerformer(x) ∧ Atomic(x) → ∃y[OperstionalPerformer(y) ∧ Implements(x,y)]] – *ResourcePerformer (x) – x is an element with stereotype <<OperationalExchange >>;* – *Atomic (x) – x is atomic and it cannot be divided;* – *OperstionalPerformer (y) - y is an element with stereotype <<ResourceExchange >>;* – *Implements(x,y) – x element implements y by Implements relationship;*	**Low** (weight=1)	**Rule Description**: Each atomic *Resource Performer* must implement at least one *Operational Performer*. **Constrained Element**: Resource Performer **Weight Justification**: The linkage between Resource Performer and Operational Performer ensures that the resource at the lower level of abstraction is used in the architecture. Resource Performers that do not implement any element are redundant. **Type**: Direct Reverse **Technical Description**: Atomic *Resource Performer* element should be linked with an *Operational Performer* element using the <<Implements>> relationship.
6.	∀x[Function(x) ∧ Atomic(x) → ∃y[OperationalActivity(y) ∧ Implements(x,y)]] – *Function (x) – x is an element with stereotype <<Function>>;* – *Atomic (x) – x is atomic and it cannot be divided;* – *OperationalActivity(y) - y is an element with stereotype <<OperationalActivity>>;* – *Implements(x,y) – x element implements y by Implements relationship;*	**Low** (weight=1)	**Rule Description**: Each atomic *Function* must implement at least one *Operational Activity*. **Constrained Element**: Function **Weight Justification**: Activity specified in the context of the *Resource Performer* reflects an *Operational Activity* in the upper layer of abstraction. The linkage between function and operational activity ensures that the function performs actions in the conduct of business activities of an enterprise. Functions that do not implement any element are redundant. **Type**: Direct Reverse **Technical Description**: Atomic *Function* element should be linked with an *Operational Activity* element using the <<Implements>> relationship.
7.	∀x[DataElement (x) → ∃y[InformationElement(y) ∧ Implements(x,y)]] – *DataElement (x) - x is an element with stereotype << DataElement >>;* – *InformationElement (y) – y is an element with stereotype << InformationElement >>;* – *Implements(x,y) – x element implements y by Implements relationship;*	**Low** (weight=1)	**Rule Description**: *Data Element* must implement at least one *Information Element*. **Constrained Element**: Data Element **Weight Justification**: Established *Data Element* reflects the *Information Element* in the upper layer of abstraction. The linkage between Data Element and Information Element ensures that Data Element is used in the architecture. Data Elements that do not implement any element are redundant. **Type**: Direct Reverse **Technical Description**: *Data Element* should be linked with an *Information Element* using the <<Implements>> relationship.
8.	∀x[ResourceExchange (x) → ∃y[OperationalExchange(y) ∧ Implements(x,y)]] – *ResourceExchange (x) - x is an element with stereotype <<ResourceExchange >>;* – *OperationalExchange (y) – y is an element with stereotype <<OperationalExchange >>;* – *Implements(x,y) – x element implements y by Implements relationship;*	**Low** (weight=1)	**Rule Description**: *Resource Exchange* must implement at least one *Operational Exchange*. **Constrained Element**: Resource Exchange **Weight Justification**: Established flow between *Resource Performers* reflects the flow between *Operational Performers* in the upper layer of abstraction. The linkage between Resource Exchange and Operational Exchange ensures that Resource Exchange is used in the architecture. Resource Exchanges that do not implement any element are redundant. **Type**: Direct Reverse **Technical Description**: *Resource Exchange* should be linked with an *Operational Exchange* using the <<Implements>> relationship.

TABLE 2. *(Continued.)* **'Operational vs Resources' validation rules**

9.	∀x[(ResourcePerformer(x) ∧ Atomic(x)) → ∃y[OperationalPerformer(y) ∧ Implements(x,y)]] → ∃z[Capability(z) ∧ Exhibits(y,z)] →[Implements(x,z)] – *ResourcePerformer (x) – x is an element with stereotype <<ResourcePerformer >>;* –*Atomic (x) – x is atomic and it cannot be divided;* –*OperationalPerformer (y) - y is an element with stereotype <<OperationalPerformer >>;* –*Implements(x,y) – x element implements y by Implements relationship;* –*Capability(x) – x is an element with stereotype <<Capability>>;* –*Exhibits(y,z) – y element implements z by Exhibits relationship;* –*Implements(x,z) – x element implements z by Implements relationship;*	**Medium** (weight=2)	**Rule Description**: When atomic *Resource Artefact* implements *Operational Performer* then that *Resource Artifact* must exhibit the same *Capability* as implemented *Operational Performer*. **Constrained Element**: Resource Performer **Weight Justification**: The linkage between Resource Artefact and Operational Performer ensures that the Operational Performer is implemented by a semantically equivalent element. Also, it is important that the related Resource Artefact and Operational Performer exhibits the same Capability because Resource Artefact reflects an Operational Performer in the upper layer of the abstraction. **Type**: Derivate **Technical Description**: Resource Artifact should be linked with an Operational Performer using the <<Implements>> relationship. Operational Performer should be linked with a Capability using the <<Exhibits>> relationship. Resource Artifact should be linked with the same Capability as Operational Performer using the <<Exhibits>> relationship.
10.	∀x[(Function (x) ∧ Atomic(x)) → ∃y[OperationalActivity (y) ∧ Implements(x,y)]] → ∃z[Capability(z) ∧ Maps(y,z)] →[Implements(x,z)] –*Function (x) – x is an element with stereotype << Function >>;* –*Atomic (x) – x is atomic and it cannot be divided;* –*OperationalActivity (y) - y is an element with stereotype << OperationalActivity >>;* –*Implements(x,y) – x element implements y by Implements relationship;* –*Capability(x) – x is an element with stereotype <<Capability>>;* –*Maps(y,z) – y element implements z by MapsToCapability relationship;* –*Implements(x,z) – x element implements z by Implements relationship;*	**Medium** (weight=2)	**Rule Description**: When atomic *Function* implements *Operational Activity* then that *Function* must map the same *Capability* as implemented Operational *Activity*. **Constrained Element**: Function **Weight Justification**: The linkage between Function and Operational Activity ensures that the Operational Activity is implemented by a semantically equivalent element. Also, it is important that the related Function and Operational Activity maps to the same Capability because Function reflects an Operational Activity in the upper layer of the abstraction. **Type**: Derivate **Technical Description**: Function should be linked with an Operational Activity using the <<Implements>> relationship. Operational Activity should be linked with a Capability using the <<MapsToCapability>> relationship. Function should be linked with the same Capability as Operational Activity using the <<MapsToCapability>> relationship.

elements, rules developed from the Resource domain point of view).

Input: Strategic domain model; Resources domain model; Predefined 'Strategic vs Resources' validation rules.

Output: Official report on the results of the 'Strategic vs Resources' architecture quality evaluation.

2) HORIZONTAL ARCHITECTURE EVALUATION

Horizontal architecture evaluation is used to determine the quality of the submitted solution itself before it can be evaluated further. This evaluation covers only the Resources domain. The Resources domain concerns the definition of solution architectures to implement operational requirements [39].

The established 'Resources' rule set (Table 4) aims to verify the quality of the Resource domain model. The set of rules contains two traceability rules based on the UAF metamodel: 1) direct (direct traceability relationship between two elements, rules developed from the Resource point of view); and 2) direct reverse (direct traceability relationship between two elements, rules developed from the Function point of view).

Input: Resources domain model; Predefined 'Resources' validation rules.

Output: Official report on the results of the 'Resources' architecture quality evaluation.

IV. EXPERIMENTAL EVALUATION

In order to confirm the suitability of the proposed validation rules for the base check, all rules were implemented using the validation-based metric technique in the MagicDraw CASE tool (Figure 4). Validation-based metrics are one of the methods to automate the evaluation process in MBSE. It is a combination of validation rules and metrics that display a set of elements of failed validation rules and provide structured information in numeric values (Table 1). A set of executable rules has been developed to verify that all three domains (Strategic, Operational, Resources) are properly modeled. For experimental evaluation, we use an implemented plug-in with the MagicDraw CASE tool. The plug-in contains validation rules and validation-based metric suites, which are developed using a structured expression.

Validation rules for vertical and horizontal evaluation of architecture have been applied to four real-world industry projects. All four projects are designed according to UAF principles. These projects are under the non-disclosure agreement and therefore only base check results are provided.

TABLE 3. 'STRATEGIC vs Resources' validation rules.

No	Validation Rule	Severity (Weight)	Description
1.	∀x[Capability(x) ∧ Atomic(x) → ∃y[ResourcePerformer(y) ∧ Exhibits(y,x)]] – *Capability(x) – x is an element with stereotype <<Capability>>;* – *Atomic (x) – x is atomic and it cannot be divided;* – *ResourcePerformer(y) - y is an element with stereotype <<ResourcePerformer>>;* – *Exhibits(y,x) – y element exhibits x by Exhibits relationship;*	**High** (weight=3)	**Rule Description**: Each *Capability* that is atomic must be exhibited by *Resource Performer.* **Constrained Element**: *Capability* **Weight Justification**: It is important that all identified capabilities are connected to resources. The linkage ensures that capability carries out a particular activity. **Type**: Direct **Technical Description**: Atomic *Capability* element should be linked with a *Resource Performer* element using the <<Exhibits>> relationship. **Resource Performer** – Resource Artifact, Capability Configuration, Known Resource, Software
2.	∀x[CapabilityConfiguration(x) ∧ Atomic(x) → ∃y[Capability(y) ∧ Exhibits(x,y)]] – *CapabilityConfiguration(x) – x is an element with stereotype <<CapabilityConfiguration>>;* – *Atomic (x) – x is atomic and it cannot be divided;* – *Capability(y) - y is an element with stereotype <<Capability>>;* – *Exhibits(x,y) – x element exhibits y by Exhibits relationship;*	**Low** (weight=1)	**Rule Description**: Each atomic *Capability Configuration* must exhibit at least one *Capability*. **Constrained Element**: *Capability Configuration* **Weight Justification**: Composite structure reflecting physical and human resources is assembled to meet capabilities. The linkage between resource and capability ensures that resource is used in the architecture. Identified resources that do not exhibit any element are redundant. **Type**: Direct Reverse **Technical Description**: Atomic Capability Configuration element should be linked with a Capability element using the <<Exhibits>> relationship.

TABLE 4. 'Resources' validation rules.

No	Validation Rule	Severity (Weight)	Description
1.	∀x[ResourcePerformer(x) ∧ Atomic(x) → ∃y[Function(y) ∧ Performs(x,y)]] – *ResourcePerformer(x) – x is an element with stereotype <<ResourcePerformer>>;* – *Atomic(x) – x is atomic and it cannot be divided;* – *Function(y) - y is an element with stereotype <<Function>>;* – *Performs(x,y) – x element performs y by IsCapableToPerform relationship;*	**Low** (weight=1)	**Rule Description**: Each atomic *Resource Performer* must perform at least one *Function*. **Constrained Element**: Resource Performer **Weight Justification**: The linkage between *Resource Performer* and *Function* ensures that resource is used in the architecture to perform a specific function. *Resource Performers* that do not perform any function are redundant. **Type**: Direct **Technical Description**: *Resource Performer* should be linked with a *Function* using the <<IsCapableToPerform>> relationship.
2.	∀x[Function(x) ∧ Atomic(x) → ∃y[ResourcePerformer(y) ∧ Performs(y,x)]] – *Function (x) – x is an element with stereotype <<Function>>;* – *Atomic(x) – x is atomic and it cannot be divided;* – *ResourcePerformer(y) - y is an element or sub with stereotype <<ResourcePerformer>>;* – *Performs(y,x) – y element performs x by IsCapableToPerform relationship;*	**Medium** (weight=2)	**Rule Description**: Each atomic *Function* must be performed by Resource Performer. **Constrained Element**: Function **Weight Justification**: It is important that all identified *Functions* are connected to *Resource Performer*. The linkage ensures that *Resource Performer* carries out a particular *Function*. *Functions* that are not performed by any *Resource Performer* are redundant. **Type**: Direct Reverse **Technical Description**: *Function* should be linked with *Resource Performer* using the <<IsCapableToPerform>> relationship.

Further in this section, we provide more details about the projects, starting with the size. Size is the number of elements in the UML-based repository, which indicates the size of the project and the expected scope of the elements to be checked. Table 5 exposes the number of elements in each project ('All elements in project' row) and the number of elements that have been inspected in accordance with the validation rules.

The following are the overall results of the proposed validation rules application to all four projects. Results are presented using pie charts (Figure 5), which contain the percentage of rules execution and violation grouped by rule sets and projects. Based on the results, it is observed that the traceability links between the required elements are most lacking or they are created incorrectly when checking the operational and resources domains. The 'Operational and Resources' set (Op vs St) of rules identified 48% of infringements compared to the total number of infringements. The fewest violations (only 12% of all violated rules) are found by checking traceability links between strategic and resources domains (St vs Rs).

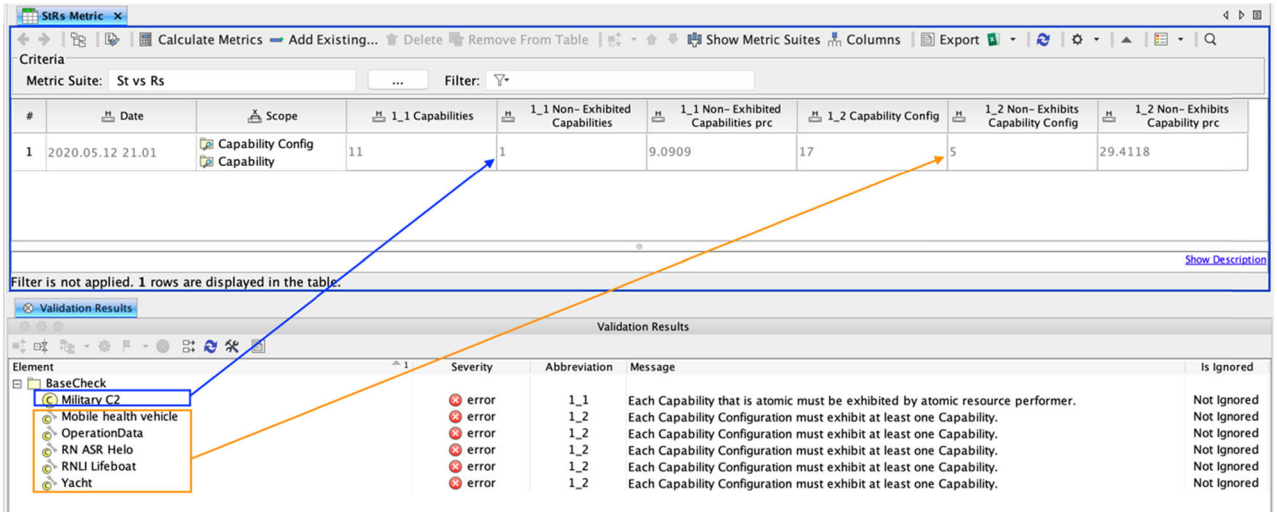

FIGURE 4. Validation-based metric designed for 'Strategic vs Resources'.

TABLE 5. Number of elements in projects.

	Project I	Project II	Project III	Project IV
Capability	51	54	45	44
Capability Configuration	57	43	48	68
Data element	118	252	120	111
Function	833	731	643	587
Information element	131	210	99	127
Operational Activity	112	233	212	116
Operational Performer	110	211	119	111
Operational exchange	212	115	212	120
Resource exchange	126	160	106	98
Resource performer	1360	1522	1432	1160
All elements in project	*1721985*	*1524177*	*1493796*	*1455295*

All predefined rules have an assigned level of severity (High, Medium, Low) according to their importance. Figure 6 shows the percentage of rules execution and violation grouped by severity level. The rules execution by grouping them according to their importance revealed that the highest number of rule violations are detected by executing the rules that are least important. The fewest violations are determined by executing the rules that are most important. This revealed that the majority of infringements occur due to rules of low importance, which do not have a significant impact on the overall quality indicator of architecture.

As the set of rules covers different sets of elements, it is important to look at the ratio of violated rules to executed rules by project (Figure 7). The closer the ratio value is to 0, the more accurately the architecture is designed and fewer violations are found. The lower value of the ratio reflects the higher level of completeness achieved in the architecture model.

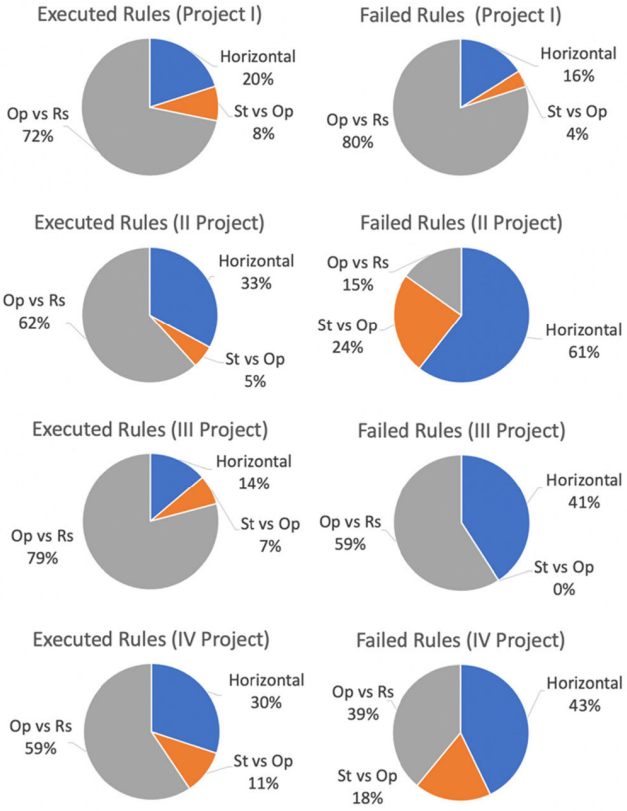

FIGURE 5. Percentage of executions and violations of the rules.

The red line in Figure 6 indicates the quality threshold. The quality threshold is calculated using the statistical mean, equation (2). Projects with a ratio value below the threshold correspond to the desired quality and can be selected as plausible alternatives for further stages of the trade study. Projects with a ratio value above the threshold are of insufficient quality and should be excluded from the trade study.

$$\bar{x} = \frac{\sum x_i}{n} = \frac{0.14 + 0.10 + 0.04 + 0.18}{4} = 0.12 \qquad (2)$$

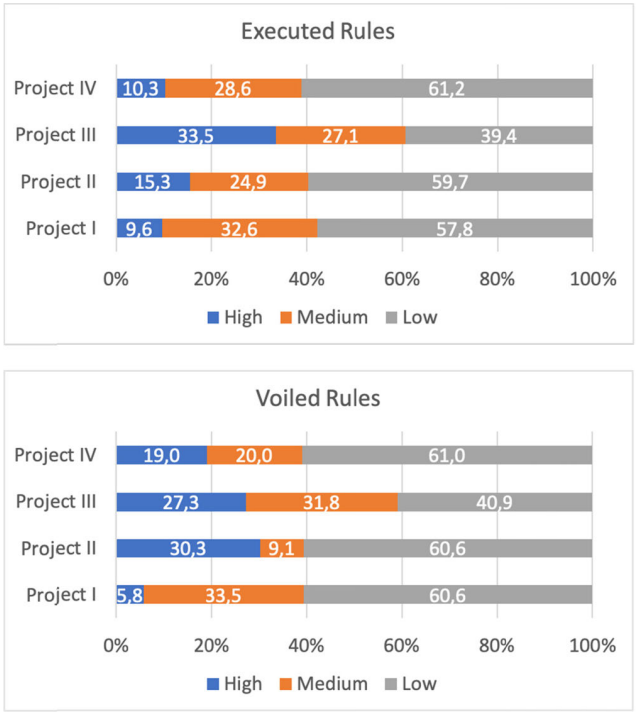

FIGURE 6. Executed and violated rules by rule severity.

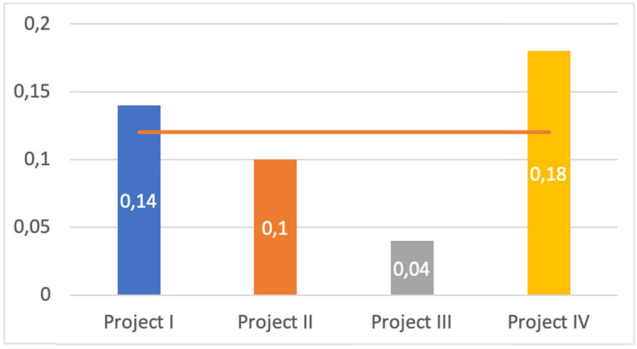

FIGURE 7. Ratio of violated rules to executed rules by project.

Based on the results, projects II and III are selected to further check the Trade Study process. Projects I and IV are not included in the trade study process due to quality discrepancies.

V. CONCLUSION

An analysis of architecture evaluation methods revealed that a scenario-based technique is the most suitable for evaluating architecture. This method provides quantitative results that allow users to compare architectures and track qualitative attributes. We also showed that the majority of the existing architecture evaluation methods are conceptual and therefore lack precise details on how to use them in a real application. To bridge this gap, we proposed a set of UAF-based validation rules for the analysis of a trade study. These rules are designed to check architectures at an early stage of a trade study, so that only quality-appropriate architectures are selected for further stages of the analysis.

The proposed set of validation rules have been implemented in the MagicDraw CASE tool. In order to confirm their eligibility, they have been applied to four real-world projects. The application of the rules disclosed the following:

- The proposed validation rules are designed in accordance with the UAF, which evolved from the Unified Profile for DoDAF and MODAF (UPDM). Therefore, we can make assume that the proposed set of validation rules can be applied to all architectures based on UAFP.
- The majority of rules violations occur due to rules of low importance (severity level - Low), which do not have a significant impact on the overall quality indicator of architecture.
- Based on the ratio of violated rules to executed rules, projects II and III meet the established quality threshold. These projects will be further evaluated in the next steps. Projects I and IV are not of sufficient quality, so they are excluded from the trade study process.

The proposed set of validation rules is one of the components of our long-term goal to support an automated trade study analysis for the SoS architecture. We aim to provide model quality checks, automation methods and techniques, which, along with UAF models, automate and accelerate the analysis of the trade study.

REFERENCES

[1] J. Bankauskaite and A. Morkevicius, "Towards an Automated UAF-based Trade Study Process for System of Systems Architecture," in *Proc. 30th Annu. INCOSE Int. Symp.*, Cape Town, South Africa, 2020, pp. 391–405.

[2] North Atlantic Treaty Organization. (Jan. 2018). *NATO Architecture Framework, Version 4*. Accessed: May 2020. [Online]. Available: https://www.nato.int/nato_static_fl2014/assets/pdf/pdf_2018_08/20180801_180801-ac322-d_2018_0002_naf_final.pdf

[3] M. M. LaSorda, J. Borky, and R. Sega, "Model-based architecture optimization for major acquisition analysis of alternatives," in *Proc. IEEE Aerosp. Conf.*, Big Sky, MT, USA, Mar. 2018, pp. 1–8.

[4] Office of Aerospace Studies. (Jul. 2016). *Analysis of Alternatives (AoA) Handbook, A Practical Guide to the Analysis of Alternatives*. Accessed: Apr. 2020. [Online]. Available: https://afacpo.com/AQDocs/AoAHandbook.pdf

[5] DoD Deputy Chief Information Officer. (Aug. 2020). *DoD Architecture Framework Version 2.02*. Accessed: Apr. 2020. [Online]. Available: https://dodcio.defense.gov/Library/DoD-Architecture-Framework/dodaf20_pes/

[6] Object Management Group. (Oct. 2019). *Unified Architecture Framework (UAF) 1.1*. Accessed: Apr. 2020. [Online]. Available: https://www.omg.org/spec/UAF/About-UAF/

[7] M. Hause, G. Bleakley, and A. Morkevicius, "Technology update on the unified architecture framework (UAF)," in *Proc. 26th Annu. INCOSE Int. Symp.*, Edinburg, U.K., 2016, pp. 1145–1160.

[8] A. Morkevicius, A. Aleksandraviciene, D. Mazeika, L. Bisikirskiene, and Z. Strolia, "MBSE Grid: A simplified SysML-based approach for modeling complex systems," in *Proc. 27th Annu. INCOSE Int. Symp.*, Adelaide, SA, Australia, 2017, pp. 136–150.

[9] OMG. (Dec. 2018). *About the Semantics of a Foundational Subset for Executable UML Models Specification, Version 1.4*. Accessed: Jun. 9, 2020. [Online]. Available: https://www.omg.org/spec/FUML/About-FUML/

[10] *ISO/IEC/IEEE International Standard—Software, Systems and Enterprise–Architecture Evaluation Framework*, Standard ISO/IEC/IEEE 42030:2019(E), 2019.

[11] N. Niu, L. Da Xu, and Z. Bi, "Enterprise information systems Architecture—Analysis and evaluation," *IEEE Trans. Ind. Informat.*, vol. 9, no. 4, pp. 2147–2154, Nov. 2013.

[12] M. A. Chauhan and M. A. Babar, "Using reference architectures for design and evaluation of Web of things systems: A case of smart homes domain," in *Managing the Web of Things: Linking the Real World to the Web*, 1st ed. San Mateo, CA, USA: Morgan Kaufmann, 2017, pp. 205–228.
[13] M. Erder and P. Pureur, "Validating the architecture," in *Continuous Architecture Sustainable: Architecture in an Agile and Cloud-Centric World*, 1st ed. San Mateo, CA, USA: Morgan Kaufmann, 2016, pp. 131–159.
[14] L. Zhu, M. Staples, and T. Nguyen, "The need for software architecture evaluation in the acquisition of software-intensive sysetms," Aerosp. Division, Defence Sci. Technol. Organisation, Fishermans Bend, VIC, Australia, Tech. Rep. DSTO-TR-2936, 2014.
[15] M. Vimaladevi and G. Zayaraz, "An investigation on quality perspective of software functional artifacts," in *Advances in Systems Analysis, Software Engineering, and High Performance Computing*, 1st ed. Hershey, PA, USA: IGI Global, 2020, pp. 109–133.
[16] A. Patidar and U. Suman, "A survey on software architecture evaluation methods," in *Proc. 2nd INDIACom*, New Delhi, India, 2015, pp. 967–972.
[17] P. Shanmugapriya and R. M. Suresh, "Software architecture evaluation methods a survey," *Int. J. Comput. Appl.*, vol. 49, no. 16, pp. 19–26, Jul. 2012.
[18] M. Efatmaneshnik, S. Shoval, and K. Joiner, "System test architecture evaluation: A probabilistic modeling approach," *IEEE Syst. J.*, vol. 13, no. 4, pp. 3651–3662, Dec. 2019.
[19] F. Brosig, P. Meier, S. Becker, A. Koziolek, H. Koziolek, and S. Kounev, "Quantitative evaluation of model-driven performance analysis and simulation of component-based architectures," *IEEE Trans. Softw. Eng.*, vol. 41, no. 2, pp. 157–175, Feb. 2015.
[20] G. Avventuroso, R. Foresti, M. Silvestri, and E. M. Frazzon, "Production paradigms for additive manufacturing systems: A simulation-based analysis," in *Proc. Int. Conf. Eng., Technol. Innov. (ICE/ITMC)*, Jun. 2017, pp. 973–981.
[21] J. Huang, J. Liang, and S. Ali, "A simulation-based optimization approach for reliability-aware service composition in edge computing," *IEEE Access*, vol. 8, pp. 50355–50366, 2020.
[22] X. Tang, E. Giacomin, G. D. Micheli, and P.-E. Gaillardon, "FPGA-SPICE: A simulation-based architecture evaluation framework for FPGAs," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 3, pp. 637–650, Mar. 2019.
[23] F. Federici, M. Micozzi, V. Muttillo, L. Pomante, and G. Valente, "Simulation-based analysis of a hardware mechanism to support isolation in mixed-criticality network on chip," in *Proc. Eur. Modeling Symp. (EMS)*, Manchester, U.K., Nov. 2017, pp. 185–190.
[24] M. Helali Moghadam, M. Saadatmand, M. Borg, M. Bohlin, and B. Lisper, "Learning-based response time analysis in real-time embedded systems: A simulation-based approach," in *Proc. IEEE/ACM 1st Int. Workshop Soft. Qual. Depend. (SQUADE)*, Gothenburg, Sweden, May 2018, pp. 21–24.
[25] Z. Zhou, Q. Zhi, S. Morisaki, and S. Yamamoto, "An evaluation of quantitative non-functional requirements assurance using ArchiMate," *IEEE Access*, vol. 8, pp. 72395–72410, 2020.
[26] M. A. Babar and I. Gorton, "Comparison of scenario-based software architecture evaluation methods," in *Proc. 11th Asia–Pacific Softw. Eng. Conf.*, Busan, South Korea, 2004, pp. 600–607.
[27] L. G. Williams and C. U. Smith, "PASASM: A method for the performance assessment of software architectures," in *Proc. 3rd Int. Workshop Softw. Perform. (WOSP)*, 2002, pp. 179–189.
[28] S. Huda, S. Alyahya, M. Mohsin Ali, S. Ahmad, J. Abawajy, H. Al-Dossari, and J. Yearwood, "A framework for software defect prediction and metric selection," *IEEE Access*, vol. 6, pp. 2844–2858, 2018.
[29] Y. U. Mshelia, S. T. Apeh, and E. Olaye, "Towards a unified process model for comprehensive software metrics suite: An introduction," in *Proc. 19th Int. Conf. Comput. Sci. Appl. (ICCSA)*, Saint Petersburg, Russia, Jul. 2019, pp. 52–56.
[30] J. D. Mireles, E. Ficke, J.-H. Cho, P. Hurley, and S. Xu, "Metrics towards measuring cyber agility," *IEEE Trans. Inf. Forensics Security*, vol. 14, no. 12, pp. 3217–3232, Dec. 2019.
[31] J. Schumann and K. Goseva-Popstojanova, "Verification and validation approaches for model-based software engineering," in *Proc. ACM/IEEE 22nd Int. Conf. Model Driven Eng. Lang. Syst. Companion (MODELS-C)*, Munich, Germany, Sep. 2019, pp. 514–518.
[32] A. Morkevicius and N. Jankevicius, "An approach: SysML-based automated requirements verification," in *Proc. IEEE Int. Symp. Syst. Eng. (ISSE)*, Rome, Italy, Sep. 2015, pp. 92–97.
[33] Z. Mengmeng, C. Honghui, Z. Xiaoxue, L. Aimin, and L. Junxian, "Functionality evaluation of system of systems architecture based on extended influence diagrams," *J. Syst. Eng. Electron.*, vol. 29, no. 3, pp. 510–518, 2018.
[34] *ISO/IEC/IEEE International Standard—Systems and Software Engineering—Vocabulary*, Standard ISO/IEC/IEEE 24765, 2017.
[35] R. Zhang, B. Song, Y. Pei, and Q. Yun, "Improved method for subsystems performance trade-off in system-of-systems oriented design of UAV swarms," *J. Syst. Eng. Electron.*, vol. 30, no. 4, pp. 720–737, Aug. 2019.
[36] N. Jaini and S. Utyuzhnikov, "Trade-off ranking method for multi-criteria decision analysis," *J Mul. Criter. Decis. Anal.*, vol. 24, nos. 3–4, pp. 121–132, 2017.
[37] National Aeronautics and Space Administration. (Jan. 2020). *NASA Systems Engineering Handbook*. Accessed: Oct. 2020. [Online]. Available: https://www.nasa.gov/connect/ebooks/nasa-systems-engineering-handbook
[38] P. W. Thompson, "The development of the concept of speed and its relationship to concepts of rate," in *The Development of Multiplicative Reasoning in the Learning of Mathematics*. Albany, NY, USA: State Univ. of New York Press, 1994, pp. 181–234.
[39] A. Morkevicius, L. Bisikirskiene, and G. Bleakley, "Using a systems of systems modeling approach for developing industrial Internet of Things applications," in *Proc. 12th Syst. Syst. Eng. Conf. (SoSE)*, Waikoloa, HI, USA, Jun. 2017, pp. 1–6.

JOVITA BANKAUSKAITE received the B.S. degree in informatics from Vilnius Gediminas Technical University, Vilnius, Lithuania, in 2016, and the M.S. degree in informatics engineering from the Kaunas University of Technology, Kaunas, Lithuania, in 2018, where she is currently pursuing the Ph.D. degree in informatics engineering. The subject of her research is the trade study between alternative system of systems architectures.

She is also a Systems Analyst at Dassault Systemes and is an OMG certified systems modeling professional (OCSMP). She has seven years of experience in software and systems engineering. Her expertise is in the field of system modeling and simulation.

AURELIJUS MORKEVICIUS received the Ph.D. degree in informatics engineering from the Kaunas University of Technology, in 2013. He is an OMG Certified UML, Systems Modeling and BPM professional. He is currently leading CATIA CoE for Cyber Systems in the EMEAR region. He is experienced with model-based systems and software engineering, as well as defense architectures (DoDAF, NAF). He works with companies such as BAE Systems, Deutsche Bahn, ZF, Ford, BMW and others. He is also the co-chairman and one of the leading architects for the current OMG UAF standard development group. In addition, he is actively involved in educational activities. He is also a lecturer, author of multiple articles, and a speaker at multiple conferences.

RIMANTAS BUTLERIS received the Ph.D. degree. He is currently the Head of the Center of Information Systems Design Technologies and a Full Professor with the Department of Information Systems, Kaunas University of Technology, Lithuania. His main areas of research are requirements engineering, semantic technologies, and business rules modeling. During his career, he has authored or coauthored more than 170 research articles and participated in more than twenty national and international research projects; in many of those he was a leading researcher.