Model-Based Enterprise Information System Architectural Design with SysML

Anargyros Tsadimas
Department of Informatics and Telematics
Harokopio University of Athens
70 El. Venizelou St, Kallithea, 17671, Athens, GREECE
email: tsadimas@hua.gr

***Abstract*—Model-based system design is commonly supported by Systems Modeling Language (SysML). When designing Enterprise Information Systems (EISs), software and network infrastructure architecture should be designed in parallel, ensuring system efficiency. Furthermore, non-functional requirements, such as performance ones, should be focused during EIS architecture design, as they play a significant role in system efficiency. The scope of this research is to provide a model-based approach for EIS architecture design, utilizing SysML as a modeling language. The system designer should be provided with alternative views, focusing on software and hardware architecture and facilitating non-functional requirements verification. Although SysML provides support for requirements specification, corresponding tools lack an automated requirements verification process. To this end, an integrated design environment is presented, not only capable of defining alternative EIS architectures, but also enabling architectural evaluation using simulation and integrating an automated non-functional requirement verification process utilizing simulation results.**

***Keywords*—Model-Based System Design, SysML, Non-functional Requirements, Requirements Verification**

I. Introduction

Systems engineering is an interdisciplinary field, focusing on the way we design and manage complex systems over their life cycles. Model-Based Systems Engineering (MBSE) is a methodology for designing systems using models. MBSE implies that system models are composed of an integrated set of representations. Tools and methodologies that support MBSE assume that the representations of behavior and structure are interconnected in a central model. SysML is intended to facilitate the application of an MBSE approach to create a cohesive and consistent model [1]. Other efforts include the ArchiMate language [2], which is based on the fact that in enterprise architecture models, coherence and overview are more important than specificity and detail.

Evidently, information system architecture design is a complex process involving different stakeholders and concerns [3], [4]. When building large-scale Information Systems (ISs), software engineering is usually focused, while the combination of software and hardware and the way it might affect the overall system performance is often neglected. Especially during system design, software architecture issues are usually dealt with as a discrete stage of the software engineering methodology applied. Requirements Engineering (RE) refers to the process of formulating, documenting and maintaining software requirements throughout all phases of software development. Requirements are categorized as functional and non-functional ones [5]. A functional requirement defines specific behavior or functions the system should provide. A Non-Functional Requirement (NFR) specifies criteria that can be used to judge the operation of a system, rather than specific behaviors.

Software architecture design, the management of related design decisions and the way it might be influenced by NFRs has been explored in the literature [6]. Though, software architecture design decisions are influenced by network infrastructure design, while NFRs, as performance requirements, can usually be satisfied by their effective combination. Thus. the identification of functional requirements, are not enough to ensure efficient IS operation. NFRs [7] (for example performance requirements) play a significant role during IS architecture design [8], since they depict the conditions under which specific system components should operate, leading to alternative design decisions. Thus they should be emphasized during information system architecture design. Visualization helps the involved stakeholders to understand and utilize the architectural design decisions [9]. Proposed architecture scenarios should be evaluated [10] and properly adjusted, to achieve an acceptable solution. Discrete architecture design tasks and corresponding stakeholders should be served by independent, interrelated views. Each view focuses on a specific design concern and is defined by a corresponding system submodel, which is part of the overall information system model serving IS architecture design. There are efforts that are not adopting SysML for the architecture description of IS. Instead, specific Architecture Description Language (ADL) are used to define the software architecture. Woods [11] explore the use of a simple, domain specific, architecture description notation in an industrial context.

The scope of this research is to propose a model-based design approach for EISs architecture based on SysML, the current modeling standard for system engineering. As NFRs, especially performance ones, play a significant role in IS architecture design, their description and verification using SysML is emphasized. To this end, SysML extensions are proposed, while the integration of system performance estimation tools is explored. SysML system models should utilize different views and explore how design decision affect system performance. Corresponding requirements should be verified in order to reach an acceptable EIS architectural solution.

Simulation is identified as an appropriate technique for the estimation of system model's performance [12]. In case of a complex system, such as information systems, simulation is

978-1-4673-6630-4/15/$31.00

more appropriate for performance measurements. Thus, the incorporation of simulation results into SysML models is also explored. Since NFRs (e.g., performance) are described by both qualitative and quantitative properties, simulation, as a quantitative method, is very effective to produce the necessary data for their verification. The use of formal methods [13] could play the role for testing quantitatively NFRs and especially performance ones if the system model could be expressed in an abstract mathematical model. These methods rely on performing appropriate mathematical analysis to contribute to the reliability and robustness of a design. Evaluating resource allocations policies could rely either on real-time measurements or running simulation on a system model.

II. Related Work

Integrated approaches for IS design are presenting in this section. These approaches are based on SysML focusing on requirements verification. The Rational Unified Process for Systems Engineering (RUP-SE) [14], targets information system engineering in Rational Unified Process and was based on Unified Modeling Language (UML). It also adopted SysML for model-driven information system design [15]. SysML block entities may be employed to describe software, hardware or workers within the system or systems under consideration, while SysML diagrams are used to describe different viewpoints. NFRs are defined during allocating software to hardware components. For example response time requirements can be defined when allocating processes to localities. SysML requirement entity, while used to depict functional requirements, is not adopted for NFR description. The description of derived NFRs is also not emphasized. Furthermore, NFR verification is not addressed within the context of RUP-SE.

Kimura et al. [16] propose SysML extensions for information system design, which are implemented within the context of a custom tool called Computer Aided System model-based System Integration (CASSI). CASSI targets information system integration, while three different design views are supported. The allocation of system components between different views is also supported. The SysML requirement entity is not used to associate requirements to system elements. Though, information system configurations defined using CASSI are evaluated using simulation to verify performance and availability requirements. This is accomplished using an external simulator. The behavior of system components is described within CASSI using sequence diagrams, transformed to simulation model by an external transformation tool. Although, NFRs can be verified, this is performed by the system designer using external tools. Evaluation results are not integrated within the SysML system model and NFR verification is not performed using it. Modeling and Analysis of Real Time and Embedded systems (MARTE) UML profile by Object Management Group (OMG) [17] supports model-based design of real-time and embedded systems. Non-Functional Properties (NFPs) are introduced to specify non-functional quantitative properties (e.g., throughput, delay, memory usage), associated to specific system design entities. MARTE profile focuses on performance and scheduling properties of real-time systems. Non-Functional constraints are introduced to define conditions the NFPs should conform to. The Value Specification Language (VSL) is used for this purpose formulating semantically well-formed algebraic and time expressions. Defining constraints with VSL enables their automated validation, verification and traceability, using external tools. Requirement association and derivation may also be depicted using NFR constraints expressed in VSL. After SysML standardization, strategies to apply SysML and MARTE profile in a complementary manner have been suggested [18], indicating the potential to combine non-functional properties and VSL expressions defined in MARTE with SysML requirements for the description of non-functional system characteristics. In any case, NFR verification is left to external tools, although NFR constraints can be useful in identifying the conditions that should be evaluated for this purpose.

Paredis et al. [19] adopt SysML for system design, focusing on embedded systems. In the proposed profile SysML requirement entity is extended with testable characteristics. Testable stereotype may be used for quantitative NFR definition. Testable requirements are associated to conditions under which the requirement is verified with the use of experiments or test cases. Verification conditions are defined as part of a test case, which in turn may be simulated using Modelica simulation language in external simulators to ensure that a design alternative satisfies related requirements [19]. To embed simulation capabilities within SysML, ModelicaML profile is used. Verification conditions associated to testable requirements are also defined in ModelicaML [20], while requirement verification is performed in an external tool. The proposed approach succeed in converting SysML system models to executable simulation models and enables visual requirement verification. One limitation of this framework is that test cases and requirement verification process are implicitly handled by a domain-specific tool, in this case vVDR. To enhance design capabilities of the system designer, requirement verification should be conducted within SysML modeling environment independently of the methods and tools adopted to evaluate alternative system designs. Furthermore, evaluation results should be incorporated within the SysML system model to be utilized by the system designer in alternative design decisions.

III. Objectives

As defined in our previous work [21], four core activities are identified in systems architecture design :

1) *Architecture Analysis* is the process of understanding the environment in which a proposed system or systems will operate and determining the requirements for the system.
2) *Architectural Design* is the process of creating an architecture. Given the requirements determined by the analysis, the current state of the design and the results of any evaluation activities, the design is created and improved.
3) *Architecture Evaluation* is the process of determining how well the current design or a portion of it satisfies the requirements derived during analysis. An evaluation can occur whenever an architect is considering a design decision, it can occur after some portion of the design has been completed, it can occur after the final design has been completed or it can occur after the system has been constructed.
4) *Architecture Evolution* is the process of maintaining and adapting an existing software architecture to meet

requirement and environmental changes. As software architecture provides a fundamental structure of a software system, its evolution and maintenance would necessarily impact its fundamental structure. As such, architecture evolution is concerned with adding new functionality as well as maintaining existing functionality and system behavior.

The scope of this PhD research is to propose a SysML-based approach to support the system designer in all the aforementioned core activities. To do so, the following objectives were identified:

1) to propose a model-based methodology based on OMG standards and a corresponding metamodel, facilitating the system designer to explore alternative design solutions and evaluate the system model before its implementation, focusing on NFRs
2) to implement all the necessary tools/plugins for EIS domain
3) to define a SysML profile in order to support the EIS architecture design
4) to fully automate requirements verification process, integrating a specific simulation environment
5) to evaluate the proposed methodology through a complex case study

This work was based on the design science research methodology [22]. Problem was identified and objectives were defined for the solution. To evaluate the proposed approach a case study where the renovation of the legacy system of a public organization was explored [23].

In the following, we shall discuss the actions taken to reach the aforementioned objectives that formulate the development of the methodology. To provide a comprehensive framework to perform all core design activities for EIS architecture, system models consist of different views serving different EIS design perspectives and activities according to IEEE 1471 standard [24], as depicted in Fig. 1. The connections between NFR view and other views show that NFRs are participating to all other views.

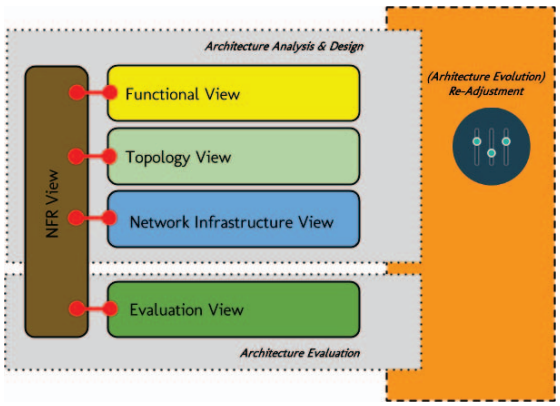

Fig. 1. MBSD EIS Architecture Views serving Core Activities

IV. Conducted Research

To achieve the defined objectives the following steps have been completed so far:

- A model-based methodology for EIS design and the corresponding metamodel have been proposed and a SysML profile for Enterprising Information Systems has been proposed [25]
- The description and the verification of NFRs and especially performance ones, through the proposition of a discrete system view, namely Evaluation view has been supported [26]
- Integration of an external simulation tool, communicating with the Evaluation view in an automated fashion has been implemented.

A. MBSD Views for EIS Architecture Design

In the following we introduce the discrete views, constituting EIS architecture models. Each of them serves specific core design activities, as depicted in Fig. 1. To define the structure of the proposed approach, we should tabulate the basic tasks identified during EIS architecture design, which are categorized as:

1) Architecture Analysis and Design: Served by design views, where a system designer defines the software and hardware architectures of the system. There are:

Functional View, consisting of software architecture description (e.g. a system-oriented view of applications). In practice, it consists of the description of functional requirements (e.g. application and data architecture, user behavior and application requirements). It also includes design decisions related to software architecture. *Roles* are used to depict the behavior of different user groups while *modules* (client & server) are application tiers (multi-layered applications) that consist of *services*. Each role *initiates* services that belong to client modules, and each service may *invoke* other services that belong to other modules, depending on the complexity of the application. *Data Entities* are used to represent portions of stored data. Entities participating in Functional view are related to entities participating in all other diagrams to implement the relations depicted in Fig. 2.

Topology View, consisting of the description of system access points. It facilitates the allocations of users , applications and data to system access points. It fills the gap between Functional view and Network Infrastructure view, facilitating the hierarchical allocation of software entities and users to hardware elements. It acts as an intermediate step, that provide information to the designer in order to facilitate him to make the appropriate allocations. Moreover, in Topology view, software component replicas can be defined, to support distributed applications. This means that many instances of roles, software components and nodes are supported. The allocation is enhanced with the definition of derived requirements that capture the load that software components and user interactions produce to hardware components.

Network Infrastructure View, consisting of the description of platform-independent distributed infrastructure (e.g. network architecture and hardware configuration) and the association of software components to network nodes (resource allocation). It refers to the aggregate network, described through a hierarchical structure comprising simple and composite *networks*. Hardware components and configurations are also defined using this view (*servers, workstations and*

network devices). Sites are allocated to networks using *Structural Allocation* relation. Each atomic network is a custom diagram, which encompasses all hardware elements that belong to that network. Elements of Functional, Topology and NFR views may also participate in Network Infrastructure view to represent inter-view relations.

NFR View consists of all NFRs that should be satisfied by entities belonging to design views. These requirements are progressively defined during model-based EIS Architecture design. Performance and availability requirements are emphasized, since they are essential in EIS architecture design. Requirements are defined in design views, each of the satisfying design entities, but also these requirements are interrelated in many ways: requirements are derived from other requirements, satisfy design entities and are verified from evaluation entities. The utilization of NFR view is not to present all requirements from design views, but relates to the distribution of the requirements to other views. NFR view bridges the gap between Design views and Evaluation view, since the NFRs are the mean through which the evaluation can be performed. Thanks to requirements verification, a system model described in design views can be evaluated against the defined requirements.

2) Architecture Evaluation and Evolution: Evaluation is consisting of these model elements that participate in the performance evaluation process along with the requirements that should be verified.

Evaluation View serves EIS architecture evaluation. Since design and NFR views are complete and valid, evaluation view collects these entities from other views that are participating in the evaluation process, initiates this process and keeps the results to validate a system against NFRs. Evaluation view is introduced to aim at defining specific EIS Architecture configurations, which should be evaluated, and storing the evaluation results of different configuration scenarios. Since EIS Architecture design process may require to evaluate and properly adjust the proposed architecture more than once, many scenarios could be evaluated. When conflicts are discovered, changes are made to the system configuration by the system architect and a new evaluation scenario is initiated by the system architect until a satisfiable solution is reached. Evaluation view encompasses the appropriate design entities along with their requirements to define which of them could be verified and informing the designer about the non-verified ones.

After identifying non-verified requirements, the system engineer may perform architecture evolution tasks, either by re-adjusting architecture decisions depicted in design view or relaxing corresponding NFRs, and initiate a new evaluation cycle.

In the corresponding EIS SysML profile, each view is represented as a discrete SysML diagram, properly extended using stereotype and constraints mechanism provided by UML. SysML Block Definition diagrams were used to depict all views, besides NFR based on SysML requirement diagram and related entities.

As depicted in Fig. 2, the aforementioned views are associated with relations such as *satisfy*, *verify*, *allocate* and *evaluate*, as supported by SysML. *Satisfy* relates system elements with requirements, *verify* relates requirements that are verified by elements from evaluation view, *allocate* relates entities from Functional or Topology view that are allocated to entities from Network Infrastructure, and *evaluate* relates entities from evaluation view that are evaluating design entities from other views.

Many stakeholders are involved in each view. Functional view serves Software Architect in order to facilitate the software architecture design. Topology view serves Network and Software Architects in order to do the resource allocation. Network Infrastructure view serves Network and Hardware Architects. They co-operate with Solutions Architects who compose the solution based on available technology. Solutions Architect has cross-domain, cross-functional and cross-industry expertise. He/she outlines solution architecture descriptions (mainly in Functional analysis), then monitors and governs the implementation. NFR view serves Network, Hardware and Software Architects to define NFRs of EIS Architecture. Evaluation view serves Network, Software and Hardware Architects to evaluate the solution that has been composed using all other views. They decide if the solution is accepted or not, so as re-adjustments to be done.

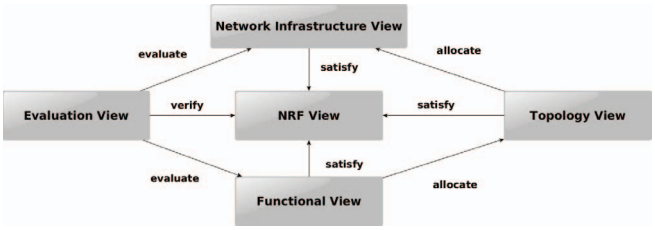

Fig. 2. EIS Architecture Design View and their Interrelations

B. Extending SysML for Performance Requirement Verification

To be able to decide if an architectural synthesis is considered efficient, criteria should be placed. NFRs, especially quantitative ones, play this role, as they provide either the ”conditions” under which the system should operate, or the ”red lines” that should be not crossed, ensuring the desired performance is provided by the proposed system design.

Evaluation View consists of multiple evaluation scenarios. An Evaluation Scenario is a set of conditions or variables which will be tested (simulated) to ensure requirements are met. As indicated in Fig. 2, an evaluation scenario is conducted to evaluate design decisions depicted in Functional and Network Infrastructure View, while its results are used to verify requirements defined in NFR View. Each evaluation scenario defines a specific solution for the system design and will be evaluated in order to accept the system model or identify which parts have failed to meet the requirements, leading to system model modification and generation of new evaluation scenarios. Regardless of the method used to perform system evaluation, evaluation entities have input properties, related to evaluated design entities, and output properties, depicting evaluation data. Based on the value of the output properties, requirements are verified or not. There are two kinds of requirements: qualitative and quantitative. In the case of quantitative requirements, the exact comparison between

arithmetic values is not always appropriate. Thus, an appropriate comparison method should be defined for a specific requirement. Requirements may also be used to depict specific behavior forced to system components. In such case, there is no point to verify them. Evaluation entities should only *conform* to them, since they specify conditions under which the system architecture design should be evaluated. Each evaluation entity is created in order to evaluate a specific EIS architecture entity and verify corresponding requirements. During system design, NFRs may also be used to depict specific behavior forced on system components (e.g., the way a traffic generator may behave under heavy traffic conditions). In this case, the corresponding evaluation entity should conform to them, providing input that could be used for the generation of the simulation model. The relation between design and evaluation entities, as well as corresponding requirements is depicted in Fig. 3. A design entity satisfies two NFRs: performance requirement (depicting system performance restrictions) and behavior requirement (depicting system behavior). Only the first requirement must be verified by evaluation entity, since the second provides input properties to the evaluation entity, indicating the conditions under which the evaluation should be done.

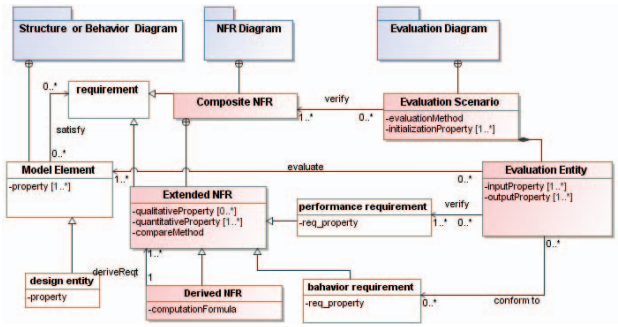

Fig. 3. Extensions to SysML Requirements and their relations to Design and Evaluation Entities

The same requirement or requirements may be verified more than once, by evolving evaluation scenarios, as the system design is re-adjusted. Evaluation data and conditions included in them should be integrated in the SysML model. The way basic SysML concepts are extended to handle NFRs for system design is summarized in Fig. 3.

C. An Integrated Framework for EIS Architecture Design

To support the proposed approach, an integrated platform was created (see Fig. 4, combing different tools facilitating EIS Architecture modeling using SysML and EIS Architecture evaluation using simulation.

As a design environment, a standard SysML modeling environment can be chosen. SysML extensions can be formally defined using such environments, while the corresponding system models may be exported in a standardized fashion in a XML format. In our case, Magicdraw was selected, where EIS SysML profile was defined and EIS related plugins were implemented using the provided Application Programming Interface (API).

The system designer only interacts with the design environment to perform system architecture design, evaluation and

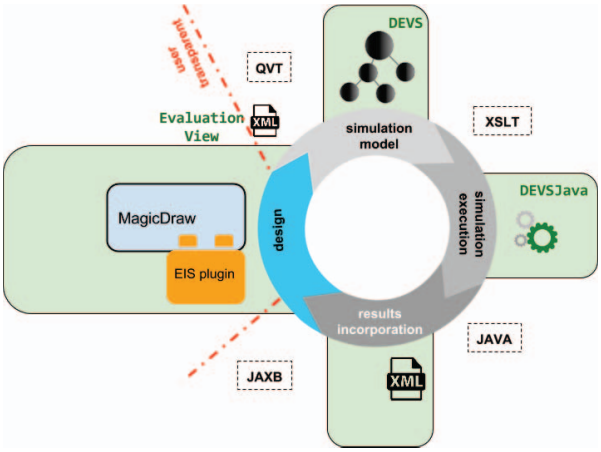

Fig. 4. An Integrated Framework for MBSD of EIS Architecture

evolution activities. The simulation process is transparent to him. Using EIS plugins, EIS architecture models are validated in terms of correctness and completeness (this is checked each time the evaluation process is started), while evaluation scenarios are auto-produced based on design view system models. Validation rules are running to ensure that system model is simulation-capable. After that, model transformation and simulation code generation are performed. Finally, the simulation results are incorporated within the Evaluation View. As a next step, the system designer is able to check whether non-functional requirements are verified. EIS plugins have been developed for both the incorporation of simulation results within Evaluation Scenarios and the automation of NFR verification process.

To simulate EIS architecture models, DEVSJava [27] simulation environment was selected, supporting discrete event simulation based on Discrete Event System Specification (DEVS). To evaluate proposed architecture models defined in hardware diagrams of evaluation scenarios, simulation code should be automatically produced. To do so, two issues should be resolved: a) the creation of EIS architecture model libraries for DEVS and b) the transformation of EIS architecture SysML models to DEVS executable simulation models. While the first one seams to be straight-forward, based on existing experience, the second one is more difficult to handle.

The transformation of the SysML model into the DEVS executable simulation model was performed based on Model Driven Architecture (MDA) principles. SysML EIS architecture models is exported in XML format, a standard XML-based representation of UML/SysML models based on Meta-Object Facility (MOF) 2.0. If a corresponding XML Metadata Interchange (XMI) representation of DEVS based MOF 2.0 was available, EIS architecture SysML-to-DEVS model transformation could be performed in a standardized fashion, using Query / View / Transformation (QVT). This is in fact, one of the reasons we choose to use DEVS for simulation purposes, as such a MOF-based representation of DEVS is available [28]. The following steps were implemented to support automated simulation code generation:

1) Export SysML model in an XMI format from MagicDraw tool

2) Define and run QVT transformation to produce the EIS architecture DEVS model from the corresponding SysML model. Have in mind that these two models are very different, as they serve different activities
3) Generate DEVSJava simulation code from DEVS XMI model, using XSLT, utilizing existing EIS library components build for DEVSJava simulator.
4) Incorporate simulation results into the EIS Architecture SysML model using the EIS plugin.

V. Conclusions and Future Work

The main objective of this research was the proposition of an Model-Based System Design (MBSD) approach for EIS architecture design using SysML. Motivated by the lack of efficient mechanisms for the verification of quantitative NFRs defined in SysML models, focus was given on the detail representation quantitative NFRs in SysML and their verification using quantitative methods. To this end, SysML was properly extended, while automated and efficient verification of SysML requirements via simulation was explored. Proposed concepts were applied in the information system domain, focusing on the design of EIS architectures, while performance requirements were focused.

The integrated framework implemented to support the proposed approach illuminates the role of models and standards towards solutions that enforce knowledge exchange and combined use of diverse proprietary tools. It targeted the facilitation of the system designer to effectively design an EIS, providing feedback about the performance of the system. In order to explore the effectiveness of the proposed approach, a complex case study concerning the renovation of the information technology infrastructure of a large-scale public organization in Greece is currently under study.

As a forthcoming research challenge, we examine the application of the proposed methodology on other system domains (e.g., transportation, communications), and to explore more types of NFRs (e.g., safety or security), as there is a lot of interest in the literature [29].

References

[1] S. Friedenthal, A. Moore, and R. Steiner, *A practical guide to SysML: the systems modeling language*. Morgan Kaufmann, 2014.

[2] M. M. Lankhorst, H. A. Proper, and H. Jonkers, "The architecture of the archimate language," in *Enterprise, Business-Process and Information Systems Modeling*. Springer, 2009, pp. 367–380.

[3] "INCOSE Handbook SE Process Model," INCOSE, September 2003, http://g2sebok.incose.org/.

[4] R. Hilliard and C. Inc, "IEEE Std 1471 and Beyond," mar 27 2001.

[5] A. Aurum and C. Wohlin, *Engineering and Managing Software Requirements*. Springer, 2005.

[6] M. A. Babar and P. Lago, "Design decisions and design rationale in software architecture," *Journal of Systems and Software*, vol. 82, no. 8, pp. 1195–1197, 2009.

[7] M. Glinz, "On non-functional Requirements." 15th IEEE International Requirements Engineering Conference, 2007.

[8] C.-W. Ho, L. Williams, and B. Robinson, "Examining the relationships between performance requirements and "not a problem" defect reports," in *RE '08: Proceedings of the 2008 16th IEEE International Requirements Engineering Conference*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 135–144.

[9] L. Lee and P. Kruchten, "Visualizing software architectural design decisions," in *ECSA*, 2008, pp. 359–362.

[10] M. H. Kacem, M. Jmaiel, A. H. Kacem, and K. Drira, "A UML-based approach for validation of software architecture descriptions," in *TEAA*, 2006, pp. 158–171.

[11] E. Woods and R. Bashroush, "Modelling large-scale information systems using adls–an industrial experience report," *Journal of Systems and Software*, vol. 99, pp. 97–108, 2015.

[12] A. Law, *Simulation modeling and analysis*, 4th ed., ser. McGraw-Hill series in industrial engineering and management science. McGraw-Hill, 2006.

[13] E. Brinksma, H. Hermanns, and J.-P. Katoen, *Lectures on Formal Methods and Performance Analysis: First EEF/Euro Summer School on Trends in Computer Science Berg en Dal, The Netherlands, July 3-7, 2000. Revised Lectures*. Springer Science & Business Media, 2001, vol. 1.

[14] M. Cantor, *Rational Unified Process for Systems Engineering, RUP SE Version 2.0, IBM Rational Software white paper*, IBM Corporation, May 2003.

[15] L. Balmelli, D. Brown, M. Cantor, and M. Mott, "Model-driven systems development," *IBM Systems Journal*, vol. 45, no. 3, pp. 569 –585, 2006.

[16] D. Kimura, T. Osaki, K. Yanoo, S. Izukura, H. Sakaki, and A. Kobayashi, "Evaluation of it systems considering characteristics as system of systems," in *System of Systems Engineering (SoSE), 2011 6th International Conference on*, june 2011, pp. 43 –48.

[17] O. M. G. Inc, "UML profile for MARTE: Modeling and analysis of real-time embedded systems specification, version 1.0," November 2009.

[18] H. Espinoza, D. Cancila, B. Selic, and S. Gérard, "Challenges in combining SysML and MARTE for model-based design of embedded systems," in *ECMDA-FA*, ser. Lecture Notes in Computer Science, vol. 5562. Springer, 2009, pp. 98–113.

[19] A. A. Kerzhner, J. M. Jobe, and C. J. J. Paredis, "A formal framework for capturing knowledge to transform structural models into analysis models," *J. Simulation*, vol. 5, no. 3, pp. 202–216, 2011.

[20] W. Schamai, P. Helle, P. Fritzson, and C. J. J. Paredis, "Virtual verification of system designs against system requirements," in *Proceedings of the 2010 international conference on Models in software engineering*, ser. MODELS'10. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 75–89.

[21] C. Haskins, *INCOSE Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities, v. 3*, International Council on Systems Engineering, June 2006, iNCOSE-TP-2003-002-03.

[22] R. Wieringa, "Design science as nested problem solving," in *Proceedings of the 4th International Conference on Design Science Research in Information Systems and Technology*, ser. DESRIST '09. New York, NY, USA: ACM, 2009, pp. 8:1–8:12.

[23] A. Tsadimas, M. Nikolaidou, and D. Anagnostopoulos, "Extending sysml to explore non-functional requirements: the case of information system design," in *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, ser. SAC '12. New York, NY, USA: ACM, 2012, pp. 1057–1062.

[24] R. Hilliard, "Ieee-std-1471-2000 recommended practice for architectural description of software-intensive systems," *IEEE, http://standards. ieee. org*, vol. 12, pp. 16–20, 2000.

[25] P. Casas, *Formal Languages for Computer Simulation: Transdisciplinary Models and Applications*. Igi Global, 2013.

[26] A. Tsadimas, M. Nikolaidou, and D. Anagnostopoulos, "Evaluating software architecture in a model-based approach for enterprise information system design," in *SHARK '10*. New York, USA: ACM, 2010, pp. 72–79.

[27] B. P. Zeigler and H. S. Sarjoughian, *Introduction to DEVS Modeling and Simulation with JAVA. DEVSJAVA Manual*, 2003.

[28] G.-D. Kapos, V. Dalakas, M. Nikolaidou, and D. Anagnostopoulos, "An integrated framework for automated simulation of sysml models using devs," *Simulation*, vol. 90, no. 6, pp. 717–744, 2014.

[29] J.-F. Pétin, D. Evrot, G. Morel, and P. Lamy, "Combining SysML and formal methods for safety requirements verification," in *22nd International Conference on Software & Systems Engineering and their Applications*, Paris, France, Dec. 2010, p. CDROM.