Simulation Modelling Practice and Theory 66 (2016) 243–259

Contents lists available at ScienceDirect

Simulation Modelling Practice and Theory

journal homepage: www.elsevier.com/locate/simpat

Simulating simulation-agnostic SysML models for enterprise information systems via DEVS

Anargyros Tsadimas*, George-Dimitrios Kapos, Vassilis Dalakas, Mara Nikolaidou, Dimosthenis Anagnostopoulos

Department of Informatics and Telematics, Harokopio University of Athens, 70 El. Venizelou St, Kallithea, 17671 Athens, Greece

ARTICLE INFO

Article history:
Received 11 January 2016
Revised 1 April 2016
Accepted 1 April 2016

Keywords:
Automated simulation code generation
Requirements verification
Model-based system design
SysML
DEVS
Non-functional requirements
Model transformations
MDA
EIS

ABSTRACT

Systems Modeling Language (SysML) is used as the modeling infrastructure in systems engineering, especially for complex systems design, independently of the system domain. Simulation is a common method to perform system model verification, during the systems development process. However, simulation code generation and execution is not integrated within the system design activity, as it is facilitated by SysML. It is either conducted as an external activity, after system design, or it affects the system design environment and practices, according to specific simulators requirements.

This paper presents how existing, simulation-agnostic SysML models from the domain of Enterprise Information System (EISs), can be transformed to executable simulation code and in addition how the simulation results can be incorporated into the source SysML model through the exploitation of Model Driven Architecture (MDA) principles and techniques. To this end, several tools and technologies are utilized, while the verification process is triggered and finalized via the system modeling environment. Adoption of MDA provides a solid, high-level infrastructure and tool availability to the proposed approach.

1. Introduction

Model-based Systems Design (MBSD) deals with the definition and exploitation of system models during most of the system development activities. Modeling Language (SysML) [1] has been proposed by the Object Management Group (OMG) to serve as a common modeling language for all these activities. It is a general purpose language, based on a variety of modeling elements grouped in different diagrams to support complex system models. SysML is a well-established language for complex system modeling and is often used in MBSD. It facilitates the description of both system structure and behavior, while design requirements may also be described in an abstract level [1]. In the work presented in this paper, emphasis is given on requirements, related to system performance, that may be verified against quantitative estimations of the system behavior [2]. A common way for deriving such estimations is simulation. Thus, there is currently strong interest in generating simulation models from SysML models [3,4].

Today, system designers utilize domain-specific profiles [5–8] defined for SysML modeling tools combined with external simulation tools to explore the performance of alternative system designs and verify corresponding requirements [9]. Usually the system designer should be familiar with both SysML models and the simulation environment and feel confident

* Corresponding author. Tel.: +30 210 9549418; fax: +30 210 9549 401.
E-mail addresses: tsadimas@hua.gr (A. Tsadimas), gdkapos@hua.gr (G.-D. Kapos), vdalakas@hua.gr (V. Dalakas), mara@hua.gr (M. Nikolaidou), dimosthe@hua.gr (D. Anagnostopoulos).

http://dx.doi.org/10.1016/j.simpat.2016.04.001
S1569-190X(16)30025-9/

in working with corresponding tools. While system design models are defined in SysML modeling tools, requirement verification and the evaluation of the proposed solution is usually performed within the simulation environment. In order to simplify the overall process, system designers should be provided with a single system model for both system design and requirement verification, while simulation model generation should be fully-automated and transparent to them. Furthermore, SysML system models should be defined, independently of the methods and tools adopted for simulation. Such a task still remains a challenge [10].

The work presented in this paper addresses these issues, by enhancing the process of SysML models evaluation and simulation code generation, without requiring the integration of simulator-specific properties in the profiles used during system design. These enhancements simplify the evaluation process, allowing the system designer to focus on the examination of the unverified requirements and, consequently, the detection of the necessary design solution re-adjustments [6]. Model Driven Architecture (MDA) [11] principles and technologies have been adopted to automate simulation code generation and integrate simulation results into the SysML modeling environment. To explore the proposed approach, we study its applicability in a specific domain, namely Enterprise Information Systems Design, using a specific simulation framework, namely Discrete Event System Specification (DEVS) [12].

Our approach provides simulation capabilities for SysML system models, without requiring the use of a simulator specific SysML profile. This renders the approach simulation-agnostic in its conception, establishment and influence to the system designer. Enterprise Information System Design is supported by the EIS SysML profile, described in [6]. No simulator-specific properties are defined in the profile. On the contrary, it focuses on different views for EIS architecture, while it enables requirement verification and the definition of evaluation scenarios. The EIS profile is examined and utilized to develop a framework that extracts information, necessary to derive executable DEVS simulation models, based on DEVS library components. DEVS was chosen, since (a) system structure is defined in both SysML and DEVS in a similar fashion, making the transformation conventional, and (b) existing tools facilitated standardized DEVS meta-model representation and enhanced simulation code generation process [4]. SysML-to-simulation model transformation needs to be defined only once for the pair of the Enterprise Information System (EIS) domain-specific SysML profile and the DEVS simulation environment, ensuring a seamless integration of SysML modeling environments and DEVS simulators for the domain of EIS. While it is applied, simulation integration is transparent to the system designer that interacts only with the SysML EIS models within a SysML modeling environment. This way, no additional simulation-related restrictions are imposed to system designers. The proposed approach for MBSD may be easily utilized in numerous domain system models, once an appropriate simulator, for the specific domain, has been chosen and the corresponding model transformation has been developed.

The rest of the paper is organized as follows: In Section 2, related work is discussed, describing relevant approaches. The pre-existing SysML profile for the domain of EIS is presented in Section 3, where specific issues and challenges that need to be addressed are also outlined. Section 4, presents a requirements verification scheme that aims at addressing these issues. It is based on MDA principles and techniques and exploits the DEVS simulation framework, in order to provide estimations about the performance of system model configurations. Section 5 provides a practical perspective of the design and evaluation process and, finally, conclusions and emerging issues from this implementation reside in Section 6.

2. Related work

SysML supports a variety of diagrams describing system composition and states, necessary to perform simulation, which are utilized by different approaches [13,14]. In most cases, SysML models defined within a modeling tool are exported in XML Metadata Interchange (XMI) format and, consequently, transformed into simulator specific models to be forwarded to the simulation environment. Depending on the nature and specific characteristics of the systems under study, there is a diversity of ways proposed to simulate SysML models, utilizing different diagrams. The authors have recently presented a comprehensive understanding of the similarities and differences of existing approaches targeting simulation code generation from SysML models in [10], where current challenges in fully automating simulation of SysML models are also identified. The most well-known of them are briefly discussed in the following.

The SysML4Modelica profile [15], endorsed by the OMG, enables the transformation of SysML models to executable Modelica simulation code. The Query/View/Transformation (QVT) standard set of languages is used for the transformation of SysML models to executable Modelica models. Since SysML profiles are based on the Unified Modeling Language (UML) extension mechanism, they can be imported in any standard UML modeling tool, such as Rational Modeler [16] or MagicDraw [17], enabling the utilization of simulation tools. In [18], focus is given on embedded systems. In the proposed profile, the SysML requirement entity is extended with testable characteristics. 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 [18]. To embed simulation capabilities within SysML, the ModelicaML profile is used. Verification conditions associated to testable requirements are also defined in ModelicaML [19], while requirement verification is performed in an external Modelica tool (MathModelica), through visual diagrams created during simulation. This effort also does not return any feedback back to the SysML modeling tool. The key limitation of this approach is that it is tailor-made to the Modelica simulation environment, in order to describe the requirements and the verification process together.

Focusing on the domain of real-time embedded systems, TEPE, a graphical expression language is introduced in [20], based on SysML parametric diagrams, representing functional and Non-functional Requirement (NFRs) in a formal, non standard way, making them amenable to automated verification. A custom methodology is used to capture requirements, while the system design is based on SysML blocks and the behavior on state machines and verification is performed via UPPAAL [21]. The effort is supported by a custom toolkit, namely TTool, interfaced to verification tools that implement reachability analysis and model-checking. DIPLODOCUS, a simulation engine targeting on System-on-Chip design, is integrated in TTool.

Simulation of discrete event systems is commonly based on system behavior described in SysML activity, sequence or state machine diagrams. In [3], system models defined in SysML are translated to be simulated using Arena simulation software. SysML models are not enriched with simulation-specific properties, while emphasis is given to system structure rather than system behavior. MDA concepts are applied to export SysML models from a SysML modeling tool and, consequently, transformed into Arena simulation models. However, the simulation models should be manually enriched with behavioral characteristics before becoming executable.

Describing the system model with a simulation tool-specific profile requires a Domain Specific Language (DSL), as the one proposed in [22], for the wafer fab simulation via SysML diagrams. Following a similar approach, a broader range of applications for discrete event simulation has been defined in [23], exploiting ontologies as a mechanism for capturing modeling knowledge in a reusable form. Finally, in [24], an adaptation of the MDA is proposed, for a finite set of potential applications, via a simulation tool-specific profile like SysML4Arena. As a proof of concept the transformation from SysML models to simulation models is achieved with ATLAS Transformation Language (ATL). A key difference of ATL with QVT is that ATL supports only unidirectional transformations, while QVT supports bidirectional transformations [25].

The DEVS formalism and simulators have been used extensively for the study and verification of systems of various domains. Some recent related efforts are described in [26–28]. Additionally, the concept of deriving executable DEVS code, using model-driven techniques and tools, has been intensively addressed, as described in a relevant survey [29]. The solid foundation of the DEVS formalism, regardless of various specific simulators, has strongly enforced the prospects for approaches focusing on high-level declarative representations of DEVS models that can be transformed into executable models for DEVS simulators.

Motivated by the advances in model-based approaches for DEVS, the wide range of domains of application and the structural similarities between SysML and DEVS, an integrated framework for automated simulation of SysML models using DEVS (DEVSys) has been proposed in [4]. In this approach, the system model should be enriched with DEVS-specific information, mainly related to systems behavior, according to a proposed SysML profile. Enriched system models are then transformed to high-level DEVS models via a QVT transformation and are further translated into executable DEVS code. However, the definition of system behavior within SysML models, although feasible, may not be as efficient as the utilization of implemented simulation components [24]. Furthermore, such simulation components may already be available for specific system model elements in the domain of application.

Recently, Systems LIfecycle Management (SLIM) [30], a commercial collaborative model-based systems engineering workspace that uses SysML as the front-end for orchestrating system engineering activities from the early stages of system development, is available from Intercax. The SysML-based system model serves as a unified, conceptual abstraction of the system, independent of the specific design and analysis tools that shall be used in the development process. It is designed to provide plugins to integrate the system model (in SysML) to a variety of design and analysis tools. Until now, only the integration of SysML and other model repositories, such as product lifecycle management (PLM) tools is implemented. Integration with MATLAB/Simulink, Mathematica and OpenModelica is offered in a variety of commercial tools, but these tools are used as math solvers and not as a verification method of an integrated SysML model in a specific domain.

ModelCenter [31], a commercial tool by Phoenix Integration, adopts a similar approach. Its intention is to integrate system models, where SysML acts as the canvas, and simulate them using specific simulation environments like Matlab. Moreover, Modelcenter is actually part of the SLIM philosophy, focusing on the automatic design optimization. However, Modelcenter focuses on SysML parametric diagrams using customized constraint blocks pointing to black-box analysis, where the systems behavior is described and is evaluated either in the Modelcenter tool or even inside the SysML tool. SysML requirements are also used for system evaluation and are treated as design constraints.

We share the vision of SLIM and ModelCenter, but at the same time we target the simulation of simulation-agnostic SysML models, e.g. models which are not enriched with any simulator-related properties. As a case study, a profile for Enterprise Information System Design is utilized, namely EIS SysML Profile [32]. It is an extension of SysML, facilitating the effective description and verification of non-functional performance requirements. The role of NFRs within SysML is exploited in ch. 8 of [33] to serve the needs of simulation. No simulation-capabilities are integrated within the profile. The proposed approach targets at standards-based integration of a DEVS simulation environment with the profile, based on the experience obtained by the development of DEVSys approach, presented in [4]. As stated in [24], since system behavior is difficult to model, our approach uses simulation tool-specific library components. The difference is that the SysML domain-specific profile is first enhanced with simulation semantics (independent of the targeting simulation environment) and then the transformation of the SysML model to the simulation tool-specific model is applied with QVT. The importance of the existence and use of simulation-specific meta-models, as well as other guidelines regarding simulation of SysML models are presented in [34].

In the work presented in the following, focus is given on providing a combination of features that may have been provided individually in the past, but they have not been jointly met, so far, due to their inherent contradicting nature and

requirements. Specifically, the presented case study provides automation of the verification process with incorporation of simulation results in the system model, while ensuring openness and simulator independence. A key aspect of the presented approach is that it enables the exploitation of existing system models defined in terms of domain-specific profiles (e.g., EIS), rather than requiring the application of simulator-specific profiles. The latter enables the application of the verification process in existing system models, instead of requiring from system designers to annotate system models beforehand. An end-to-end implementation is illustrated, where each step is analytically described to help potential future application in other domains.

Compared to ModelCenter [31], our approach is based on extending SysML requirement entity, which eliminates the need to perform model analysis in order to calculate derived properties of complex requirements. Moreover, system behavior is described using specific requirements and is evaluated using simulation library components. Another major difference is that in our approach QVT is used for model transformation in contrast to analysis modeling that is used in the case of Modelcenter.

3. Problem statement: verifying non-functional requirements in EIS design

Non-functional requirements specification and management has been addressed in [2,35] for the domain of EIS. In the context of the above mentioned work, emphasis was given on requirements analysis, management and organization, ensuring that requirements are verifiable. This was mainly achieved by introducing quantified properties, regarding performance requirements. However, effective utilization of this kind of analysis presupposes the availability of estimated, quantified performance indications.

This is commonly achieved through external development and execution of simulation models, which is a manual or semi-automated procedure. An alternate approach is the enrichment of the system model with detailed simulation properties, enabling simulation code generation [4,15]. Both approaches require extensive post-processing of the system model, either in external analysis environments or within the modeling tool.

In this paper we aim at automating executable simulation code generation, avoiding both manual post-processing and simulator-specific bias on the system modeling activity. This enables the application of the approach to models that have been designed, independently of the approach. In this sense, the domain-specific SysML profile for EIS has been developed in advance and independently of the proposed approach and is briefly described here. It focuses on software and hardware architecture design, emphasizing non functional requirements specification.

In [36], the authors proposed a methodology focusing on EIS architecture design. In this context and according to the basic processes that are identified during the design activity, solution synthesis encompasses *Functionality, Topology* and *Network Infrastructure* definitions. Functionality Definition focuses on software architecture design, Topology Definition on software allocation process and Network Infrastructure Definition on hardware architecture design. For each of these concerns, a corresponding view is defined to explore non functional requirements and related design decisions. The system designer, concerned with software architecture design and software allocation, is served by Functional and Topology views to contribute in the construction of EIS architecture. In a similar fashion, the designer is contributing to hardware configuration using Network Infrastructure view.

A complementary view, called *Evaluation* view was proposed in [6,33], to serve system evaluation activity and manage evaluation results and requirement verification. This view incorporates entities from other views and stores all the required attributes for each model element, in order to facilitate the requirements verification process. The evaluation view is based on SysML block and Internal block definition diagrams. Behavior is modeled with corresponding behavior requirements that are the traffic generators on the system, so behavior diagrams like activity or state machines are not used in this approach.

The aforementioned views are associated with relations such as *satisfy, verify, allocate* and *evaluate. Satisfy* relates system elements with requirements, *verify* relates requirements that are verified by elements from Evaluation view, *allocate* relates entities from Functional or Topology views that are allocated to Network Infrastructure view entities, and *evaluate* relates entities from Evaluation view to elements, being evaluated, from other views (design views).

In the case of EIS architecture design, the evaluation view facilitates:

1. the definition of the conditions under which the system will be evaluated;
2. the incorporation of the evaluation results;
3. the verification of requirements, informing the system designers for inconsistencies.

To this end, the Evaluation view consists of *evaluation scenarios*. 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.

Evaluation scenarios comprise of evaluation entities used to evaluate design entities. Moreover, evaluation entities verify specific requirements associated with them. 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 specific requirements. Requirements may also be used to

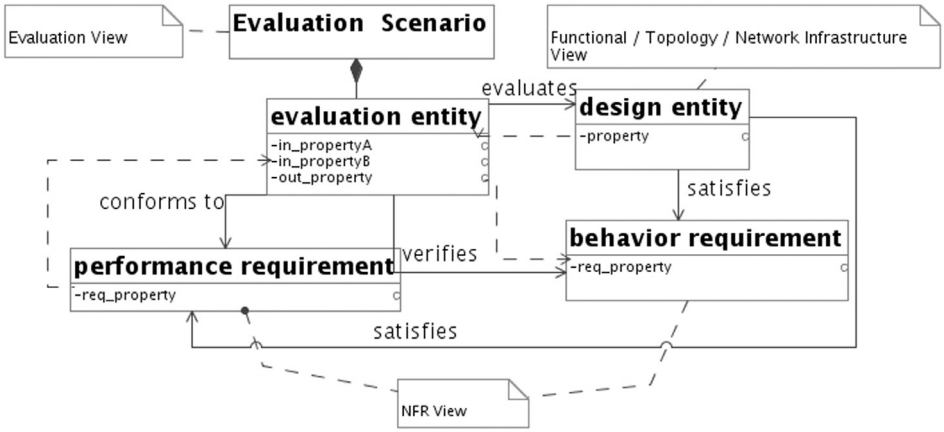

Fig. 1. Interrelating EIS performance requirements, design entities and evaluation entities.

depict specific behavior forced to system components. In such case, there is no point to verify them. Evaluation entities *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 conforms 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. 1. 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 an evaluation entity, since the second provides input properties to the evaluation entity, indicating the conditions under which the evaluation should be done.

Each evaluation scenario consists of two sub-views (diagrams), focusing on software and hardware design, respectively. The entities participating in software evaluation diagrams correspond to entities from the functional view and are used to define the behavior of the software components during the evaluation of the proposed EIS architecture design. These entities are used to evaluate the corresponding Functional view entities and to verify the related requirements. Hardware evaluation diagram entities correspond to entities from Topology and Network Infrastructure views, and are used to initialize a corresponding simulation model instance and evaluate the design entity and verify the corresponding requirements.

To support the EIS profile, a plug-in was developed for the MagicDraw UML tool [32]. This plugin that handles model constraints and embed the desired functionality in MagicDraw, was slightly modified here, in order to apply the functionality presented in this paper (e.g., the incorporation of the simulation results, described in Section 4.4).

Provided the availability of the described infrastructure for performance requirements modeling and processing, a set of issues and challenges emerge, while considering automated simulation code generation and execution. Since simulation code is not supposed to be produced through manual post-processing of the system model, an alternate, appropriate process should be introduced. The need for simulation experts is still evident. However, their role in this process needs to be identified, while other kind of expertise may be required, as well. Selection criteria and restrictions, regarding simulators that can be integrated in such an approach need to be addressed. The infrastructure for deriving executable simulation models from system models should also be specified and selected. Finally, management of the simulation results, including their incorporation in the system model, must be achieved, in order to enable automated verification of performance requirements.

4. Simulating EIS SysML models

4.1. Stakeholders, process and artifacts

System designers may benefit from the existence of an automated verification facility. However, this facility can be effectively provided only in the context of a proper process, using a respective, implemented framework. Development of such a framework would involve more stakeholders:

- the domain experts, that would select appropriate modeling tools and provide information regarding the domain to simulation experts,
- the simulation experts, that would select an appropriate simulation methodology and find or implement required simulation components,

Table 1
Stakeholders and their involvement.

Activity	Stakeholder	Software Used
Setup Phase		
Profile Definition	Domain Expert Modeling Expert	Modeling Tool
Simulation Components Selection or Development	Simulation Expert	Simulation Framework Library Components
System to Simulation Transformation Definition	Modeling Expert Domain Expert Simulation Expert	Model Transformation Tool
Design Time Phase		
System Design	System Designer	Modeling Tool
Model Validation	(Automated)	Modeling Tool
NFRs Verification	(Automated)	Modeling Tool Model Transformation Tool Simulator
Model Adjustment	System Designer	Modeling Tool

- the modeling experts, that would advise domain and simulation experts and implement system-to-simulation model transformations,
- the system designer, that would design system models and receive verification results.

Moreover, as SysML models in general cannot be directly simulated [3,34] and different types of system models are better simulated with specific simulation environments [37], the simulation environment should be customized, depending on the system domain (e.g., EIS). In order to utilize model transformation tools, the simulation environment should support input in a model format (according to a meta-model), preferably defined in terms of Meta-Object Facility (MOF).

Here, QVT is used, complementary to the MOF, in order to improve the system-to-simulation model transformation procedure and the quality of produced simulation model, compared to any kind of ad-hoc implementation of the transformation. First, QVT transformations are based on the source and target meta-models, ensuring syntactical correctness of the generated simulation model. Second, the transformation is defined as a series of relations. Each QVT relation identifies respective structural and data elements between parts of the source and target meta-models, that are applied when specific preconditions are met. Third, the fact that both source and target meta-models are MOF-based, facilitates the definition of the transformation, as conceptually equivalent infrastructural elements are used in both sides of the transformation. Forth, the transformation is defined only once per domain and simulation framework by modeling, domain and simulation experts. This justifies extensive and thorough testing of the transformation, during its development. Additionally, any unidentified shortcomings of the transformation will arise and be addressed, during its expected long-term and multiple usages.

Nevertheless, approaches focusing on various aspects of model transformation correctness [38] could be applied to effectively identify deficiencies in the transformation. Finally, the use of existing simulation library components denotes that system-to-simulation model transformation is limited to structural aspects of the system model, rather than also dealing with behavioral aspects, that are more cumbersome to validate.

Table 1 summarizes the stakeholder's involvement during *setup* and *design time* phases. The *setup* phase refers to the development activities that need to complete, before simulation of any given system model may be executed. The table clearly illustrates that advanced domain, simulation and modeling expertise is not only required during the setup phase, but also captured in products of this phase, like the domain-specific profile, the simulation library components and the system model to simulation model transformation. During *system design time*, designers enjoy the enhanced model NFR verification experience from within the modeling tool, for any given system model.

Issues concerning the specific actions of these stakeholders and the way their activities are verified is further analyzed. A specific domain profile, called EIS profile, was proposed in [39]. It enables the definition of NFRs and ensures the validity of the defined models. The EIS profile defines different views to serve the involved stakeholders (see Section 3).

The system designer (EIS engineer) is supposed to use a single UML/SysML modeling tool (e.g. MagicDraw). The EIS profile is utilized for the definition of valid system models inside the modeling tool. Fig. 2 depicts how an EIS model is transformed to its respective complex DEVS simulation model that, in its turn is simulated in DEVS simulators, utilizing a set of domain specific DEVS simulation components.

Ideally, system models would be descriptive enough, to enable simulation models generation. However, as proposed in [8,40] this can only be ensured with the use and application of simulation-specific profiles. Alternatively, existing executable simulation components could be initialized and composed according to the structure and specific attributes of the domain model elements. The designation and structuring of the key concepts of the adopted perspective leads to specific steps and preconditions that should be met by the selected software tools. Specifically:

1. The elements of the system model that will be exploited for the generation of the simulation model and the simulation execution conditions (time, runs, etc.) must be identified. The system model to simulation model transformation will be based on specific parts of the system model that affect the system's non-functional behavior. Despite their significance, these cannot always be easily detected. Additionally, system model attributes that may determine simulation execution parameters must be located. This task is performed by the domain expert in cooperation with a simulation expert.

Fig. 2. EIS verification using DEVS.

2. Depending on the domain systems' nature, a class of simulation frameworks is appropriate, e.g. for continuous or discrete event systems. Also, provision of an input MOF meta-model for a simulation framework enables utilization of standard transformation tools. This task is performed by the simulation expert in cooperation with the domain expert.
3. Existing domain specific simulation library components should be utilized. Within a domain, the behavior of several simulation components, corresponding to domain entities is usually defined using parameters. The transformation should emphasize on systems' structure and interconnections and provide parameters for the initialization of the simulation components. This task is performed by the modeling expert in cooperation with the domain expert.
4. Missing domain-specific, required simulation library components must be developed. In the vein of the previous point and although simulation components' behavior could derive from the transformation of the system model, it is more efficient and straightforward to develop the required simulation components. This way, simulator-specific details are not introduced in the system model and the transformation remains simple. This task is performed by the simulation expert.
5. Validation rules on the model elements should be defined in a declarative and quantifiable manner. This enables automation of the requirements verification process, leading to the indication of the non-verified requirements. This task is performed by the domain expert and the modeling expert.

In the above mentioned preconditions neither the domain (EIS), nor the specific simulation framework (DEVS) are referenced, outlining the generic nature of the underlying approach. The work presented in this paper is an application of this approach for the domain of EIS, with DEVS as the selected simulation framework. As instructed by the approach, the main prerequisite is a domain-specific, simulation-agnostic profile (not influenced by a simulation framework) that will be analyzed to identify the parts that determine the systems performance. An appropriate simulation framework and simulation library components should then be selected. Thus, the approach does not restrict to specific domains or simulation frameworks. However, due to compatibility issues, specific domains would be simulated only with appropriate simulation frameworks. This process is facilitated by the introduction of the abstraction layer of the simulation meta-model, allowing high-level conceptual exploration of compatibility prospects.

Nevertheless, in order to actually apply and use the approach in a specific domain and a selected simulation framework, there are domain-specific and simulation-specific tasks that are required to be performed first. These are the *setup phase*

activities of Table 1 that conclude in the determination of a domain-specific profile, a transformation of system models to simulation models and a set of simulation library components. Although there is no initial restriction on the domain or simulation framework (generic approach), once these have been selected, there are very specific artifacts that need to be developed, so that the application of the approach becomes operable.

The *evaluation view* of the EIS profile meets preconditions (1) and (5), mentioned earlier. This view defines the system model attributes that determine the simulation execution parameters. Moreover, validation rules on system model elements are defined, so as to notify the designer in case of non-verified requirements. Since DEVS is a discrete event simulation framework, suitable for checking the performance requirements of information systems, it was selected to enable the simulation of EIS models meeting precondition (2). Hence, precondition (4) was satisfied with the development of specific EIS simulation library components in terms of DEVSJava [41] (i.e., a DEVS simulator, provided as a Java class library). The corresponding simulation components were exploited in the transformation from EIS SysML models to EIS simulation models, as precondition (3) suggests.

Moreover, a set of QVT relations has been defined to provide a standards-based transformation of EIS models to the respective DEVS models. The generated DEVS model complies to an extended version of the DEVS meta-model presented in [42], that manages available simulation components [4]. It contains structural information as well as initialization parameters for existing EIS DEVS components and simulation execution parameters. In the next step, simulation is executed and the results are represented as a *results model*. The results model is then incorporated into the elements of the EIS model by the corresponding MagicDraw plugin. Incorporation of generated simulation results in the EIS model, enables validation process execution and identification of non-verified requirements, which are prompted to the EIS designer. Estimated performance of the EIS components is compared against the defined requirements in MagicDraw. The system designer is involved again in this repetitive process, only if adjustments to the EIS model are required for the elimination of non-verified requirements.

In the following, the main issues, regarding the simulation of EIS models, are explained in detail.

4.2. Simulation framework

In the case of EIS, where users and software services generate requests on other services, simulation frameworks supporting discrete time simulation are well suited. The requests generate specific traffic on the network and processing and I/O load on the site of service. Stochastic functions may be used to define non-deterministic behavior of specific parts of the system, like time handling and type of requests made by users.

Therefore, DEVS is appropriate for EIS simulation by definition. Modelica is better suited for simulating systems with continuous behavior, but it can also be used for discrete time simulation. Regarding the requirement for MDA compliance, the selected simulation framework must provide a standards-based specification for input simulation models, i.e. a MOF meta-model. This enables the definition and execution of standards-based transformations of EIS models (UML meta-model) to simulation models. Modelica and DEVS simulation frameworks both provide MOF meta-models for simulation models specification, in the notion described above. As authors already had experience on simulating SysML models with DEVS [33,34], this simulation framework was selected.

In the context of our previous research on simulating SysML models with DEVS, focus was given on defining behavior of DEVS atomic models and combining them in DEVS coupled models, rather than using existing simulation components. Therefore, the DEVS meta-model, proposed there, did not feature provisioning for this case. However, in the case of EIS, models are composed of large amounts of interconnected components of specific types (e.g. processing nodes, network nodes, services, etc.). As simplification of the verification activity is our key goal, there is no need for EIS engineers to define each component's behavior, using yet another (than the EIS) simulation-specific profile. Therefore, the required simulation components where analyzed and implemented during the setup phase of the approach for EIS. On the other hand, initialization, composition and interconnection of the components emerge from the given EIS model, during the verification execution.

The described conditions clearly indicate the need for simple and effective utilization of the EIS attributes and composition information, in combination with simulation library components. In a MDA approach, like the one presented in this paper, the simulation meta-model is the fundamental infrastructure for any kind of reference in regard with simulation model definition and capabilities. Therefore, the simulation meta-model should provide facilities for the initialization and use of simulation library components.

Such features were not available in the previous version of the DEVS meta-model. Component references of DEVS coupled components could only refer to other DEVS components that were also defined in the simulation model. In order to enable the utilization of simulation library components, the DEVS meta-model has been extended, as illustrated in Fig. 3.

In the revised version of the meta-model, component references, contained in the component reference list of a DEVS coupled component, may also refer to existing simulation library components (LIBRARY_COMPONENT). In this case, initialization parameters (INIT_PARAMS) can be specified in the simulation model, to accommodate the initialization of the library components, during the simulation execution. These parameters might be single values (VALUE_INIT_PARAM), multiple values (ARRAY_INIT_PARAM) or even other simulation library components (COMPONENT_INIT_PARAM). The last case was included to enable the definition of higher-level simulation components that comprise of more detailed components. Thus, the DEVS meta-model was extended, so that simulation library components may be initialized and utilized in any arbitrary way.

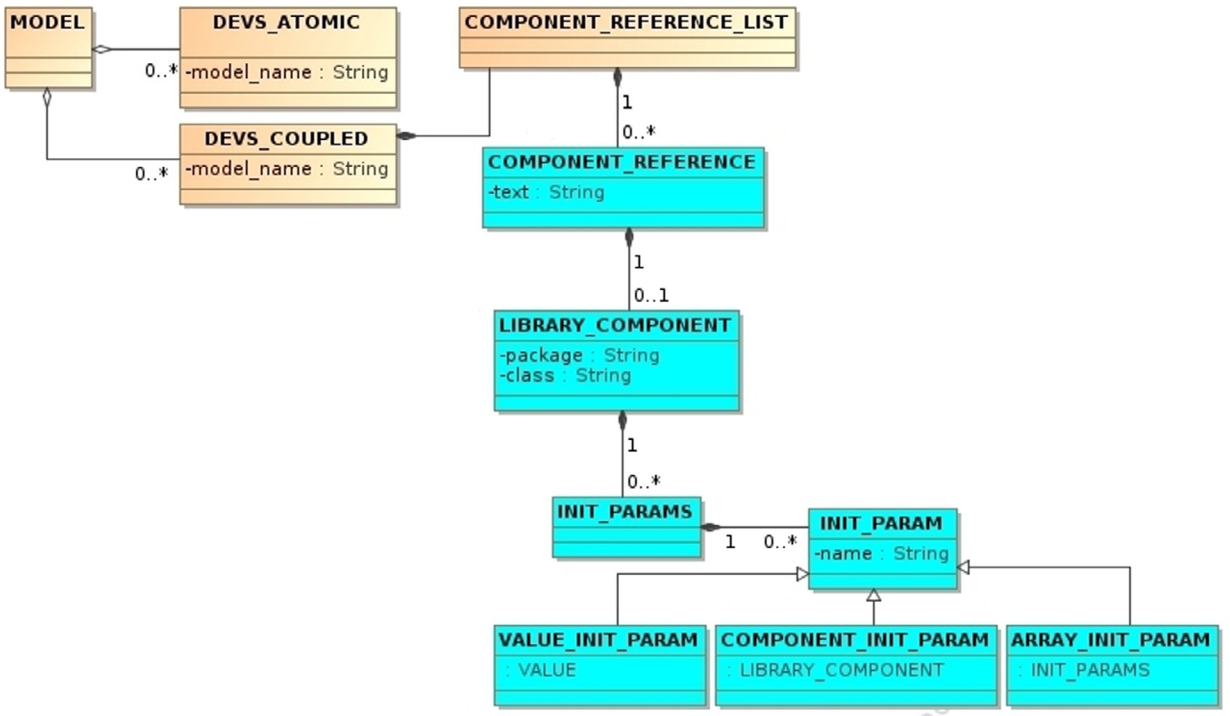

Fig. 3. DEVS Meta-model extension (cyan blocks).

DEVS models are rendered executable, once a DEVS execution environment that supports the meta-model is available. Since the DEVS meta-model has been revised, no such environment was available. Therefore, DEVSJava was selected and extended with a transformation layer that translates complex DEVS models to DEVSJava executable code. As expected, this transformation layer identifies the referenced simulation library components and their initialization parameters, to properly integrate them in the DEVSJava code that will be executed.

Given that the required simulation components are considered to be available DEVSJava components, the essential information contained in a DEVS model is the initialization, interconnection and configuration of such components. Therefore, as far as the DEVS meta-model compatible environment is concerned, we decided to implement a transformation of DEVS models (in XMI format) to the respective DEVSJava configuration class that instantiates and configures all simulation library DEVSJava components, forming, this way, the executable DEVSJava code. This transformation was defined using EXtensible Stylesheet Language Transformations (XSLT), as it is basically a syntactic transformation that exploits initialization and interconnection information in the DEVS model and creates the respective Java declarations and statements. This way, the generated DEVS simulation code can be executed in a DEVSJava environment, after an automated transformation has been executed.

4.3. Executable simulation model generation

We consider that models conforming to the DEVS meta-model, as described in Section 4.2, are valid executable simulation models. Thus, executable simulation model generation requires the transformation of EIS models to the respective DEVS models. Therefore, existing EIS models that have been developed in SysML modeling tools, according to the EIS domain-specific profile, are the source of this transformation. From the information included in such simulation-agnostic models, the respective simulation models should derive, capturing their structure and initialization information for appropriate existing simulation library components.

In order to develop this transformation, the structure and relationships of the EIS models were analyzed. The main model entities that affect overall performance were identified in both Hardware and Software Evaluation Diagrams (that are included in the Evaluation view), like Eval-Network, Eval-Node etc. Although, for EIS design purposes, it is better to define different aspects of EIS elements using different diagrams/views, the simulation model should capture the structure and attributes of a given concrete system each time.

Additionally, verification and effectiveness of the simulation largely depends on the simplicity of the simulation model. Apparently, model structure and model element attributes and dependencies exist and can be used in the transformation independently of the diagrams they appear in, which are more useful for organization/presentation purposes. Therefore,

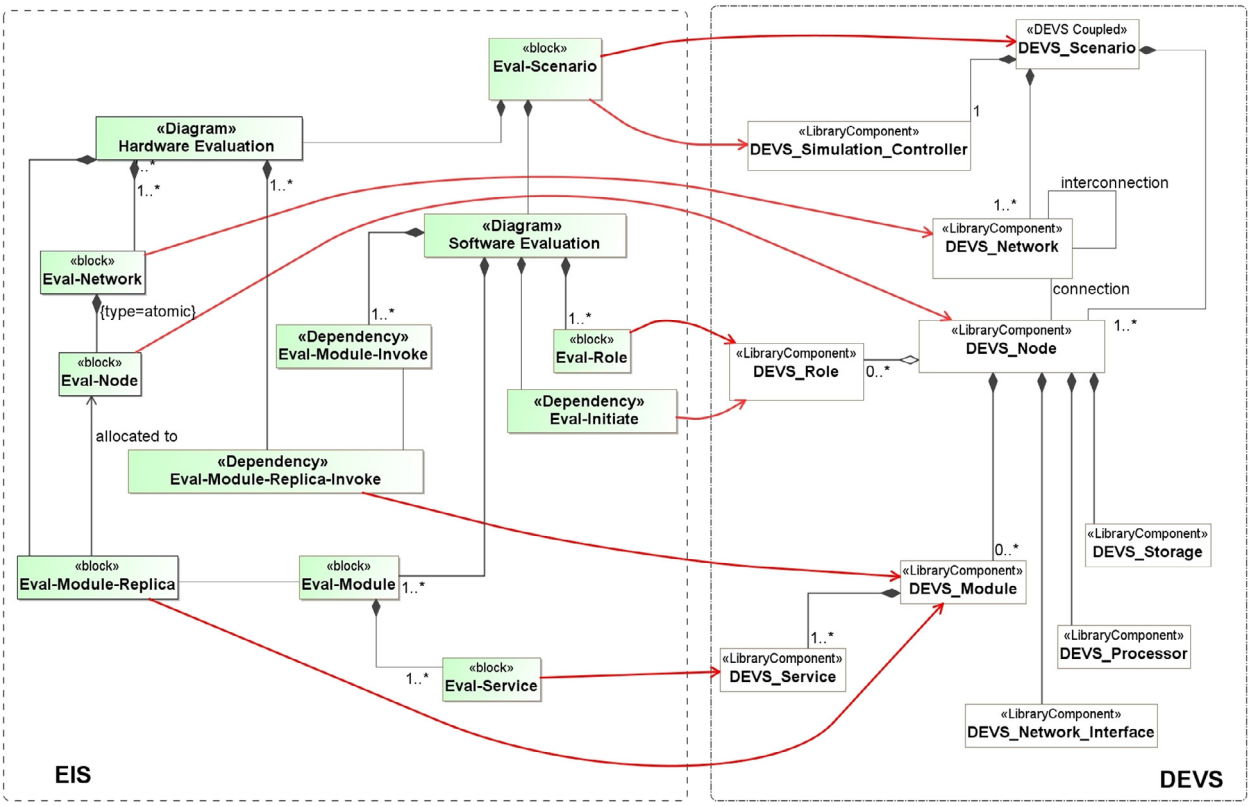

Fig. 4. Outline of the EIS to DEVS model transformation.

a set of simulation components were defined as an equivalent to EIS software and hardware elements, as they could be combined in the context of a given evaluation scenario. This is coarsely depicted in Fig. 4, where EIS model elements reside on the left side of the figure and the respective DEVS model elements on the right. The figure illustrates the high-level patterns of the conceptual mapping between EIS model elements and DEVS model elements. However, these patterns can be applied on large and complex configurations of EIS, with several hundreds of nodes. It is clearly illustrated that simulation elements usually derive from the combination of more than one EIS model elements, with the exception of DEVS_Simulation_Controller. The latter was decided to be defined independently from the DEVS_Scenario, in order to separate simulation model structure from simulation execution context, like simulation duration or number of simulation executions.

The *EIS* part of Fig. 4 includes the main entities of the Hardware Evaluation Diagram (Eval-Network, Eval-Node, Eval-Module-Replica-Invoke), the main entities of the Software Evaluation Diagram (Eval-Role, Eval-Module, Eval-Service) and their associations and dependencies. Also, the Eval-Scenario block provides information regarding the context of the evaluation that is used for the generation of the respective, top-most DEVS_Scenario and the DEVS_Simu-lation_Controller elements. These EIS entities are utilized for the generation of the respective DEVS elements (DEVS_Network, DEVS_Node, etc.), while associations and dependencies mostly indicate how DEVS model elements should be interconnected. In EIS, elements are organized under diagrams, which are combined in a scenario. On the contrary, DEVS elements are organized in a more strict hierarchy. The DEVS_Scenario is the root of the model containing the DEVS_Simulation_Controller, a set of interconnected DEVS_Network elements and a set of DEVS_Nodes. Each DEVS_Node is composed of a DEVS_Processor, a DEVS_Storage, a DEVS_Network_Interface and a set of DEVS_Modules, containing sets of DEVS_Services. Additionally, the DEVS_Roles using each node are specified, as well as the DEVS_Network, where each node belongs.

In a lower level of this transformation scheme, EIS entities were further analyzed to identify their key attributes that determine the performance of the system. Equivalent attributes were defined in the respective simulation elements. The transformation handles their proper initialization, based on the values of the respective attributes of EIS elements. The possible entity interconnection schemes were also examined. Additional information derived from the combination with other EIS model elements, like the Eval-Initiate dependencies that indicate which services are initiated by each user role. Actually, a large set of attributes and associations of the EIS model, that cannot be depicted in the high level representation of Fig. 4, are utilized in the implemented transformation.

DEVS_Scenario, DEVS_Node and DEVS_Module were implemented as DEVS Coupled components that are composed of other components. All other leaf elements were implemented as DEVS Atomic components with the expected behavior. The

```
transformation eis2devsMM(eis:uml, devs:Devs) {
top relation eisScenario2DevsModel {
 scenarioName: String;
 checkonly domain eis scenario : uml::Class { name = scenarioName };
 enforce domain devs model : Devs::MODEL {
  DEVS_COUPLED = devsCoupled : Devs::DEVS_COUPLED {
   MODEL_NAME = modelName : Devs::T_Model_Name { text = scenarioName ↩
       },
   COMPONENT_REFERENCE_LIST =
    componentReferenceList : Devs::T_Component_Reference_List { },
   INTERNAL_COUPLING = internalCoupling : Devs::T_Internal_Coupling {↩
       } } };
 when { scenario.getAppliedStereotype('EvaluationView::Evaluation ↩
     Scenario')->notEmpty(); }
 where {
  eisSimulationAttributes2ComponentReference(scenario, ↩
      componentReferenceList,
    internalCoupling);
  eisEvalNetwork2ComponentReference(scenario,componentReferenceList,↩
      internalCoupling);
  eisIncludes2InternalCoupling(scenario.getModel(),internalCoupling);
  eisIncludesRev2InternalCoupling(scenario.getModel(),↩
      internalCoupling);
  eisEvalPTPConnection2InternalCoupling(scenario.getModel(),↩
      internalCoupling); } }
 ...
}
```

Listing 1. EIS to DEVS QVT transformation.

interested reader may refer to the source code of the aforementioned library components, in [43]. Each library component is a Java class that extends DEVSJava basic classes ViewableAtomic or ViewableDigraph. Simulation execution can be inspected via SimView, a free GUI framework for DEVSJava [44]. In ViewableAtomic classes, the behavior of the simulation component is defined as determined by the component state and affected by the reaction to external events. In ViewableDigraph classes, composite simulation components, containing other interconnected components, are defined. The behavior of ViewableDigraph components is defined by the composition of the behaviors of the contained components. Implementation details of the DEVS simulation library components for EIS, are not further discussed, as they are beyond the scope of this paper.

The fact that both meta-models (SysML and DEVS) are MOF-based, enables the use of standard transformation languages, like QVT. Therefore, the appropriate QVT relations were defined for the generation and interconnection of DEVS model elements from the respective EIS model elements. The first relation of the EIS to DEVS QVT transformation is displayed in Listing 1.

Executable simulation models contain all required information, indicating the importance of their automated generation from system models, without the need for human interference. A part of a generated DEVS simulation model is presented in Listing 2, where a DEVS Coupled scenario contains a SimController DEVS library component and a Network DEVS library component. Initialization parameters, derived from the EIS SysML model, are also included.

4.4. Simulation results incorporation

Having generated the executable simulation model, simulation may be executed and the simulation results should be incorporated in the EIS model. Thus, requirement verification through the profile, independent from the simulation environment is possible.

In order to be able to use simulation results and import them in the EIS model via standards-based model manipulation approaches, they should be provided in a standard representation, i.e. according to a MOF meta-model. A simple meta-model for the representation of performance estimation, was defined for that purpose, as illustrated in Fig. 5. For each EIS model element there are two properties recorded: one holding information related with the identification of the model elements, such as the *name*, the *stereotype name* and a *unique identifier* and the second holding information related with the simulation results for that model element, such as a name-value pair for each attribute of the model element that will be imported to the system model.

A part of the Extensible Markup Language (XML) representation of produced simulation results is listed in Listing 3.

Having the simulation results imported in the system model, the only thing that the system designer should do is to run validation rules. Validation rules are utilized to indicate the evaluation entities associated to non-verified requirements. Evaluation entities not verifying a model constraint are marked with red color, and the designer is able to identify the non-verified requirement, by clicking on them.

Having such a representation, the incorporation renders to a one step procedure. The plugin that was referred to Section 3 was extended in order to support the incorporation of the simulation results into the system model, loaded in the MagicDraw design tool. The plugin uses the Java Architecture for XML Binding (JAXB) framework [45] in order to map

```
<?xml version="1.0" encoding="UTF-8"?>
<Devs:MODEL xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:Devs="urn:DEVS_MM.ecore"
xsi:schemaLocation="urn:DEVS_MM.ecore platform:/resource/Eis2DevsMM/↩
    metamodel/DEVS_MM.ecore">
  <DEVS_COUPLED>
    <MODEL_NAME text="aScenario"/>
    <COMPONENT_REFERENCE_LIST>
      <COMPONENT_REFERENCE xsi:type="Devs:T_Component_Reference" text=↩
          "SimulationController">
      <LIBRARY_COMPONENT class="SimController" package="eis.library">
          <INIT_PARAMS>
            <INIT_PARAM xsi:type="Devs:T_Value_Init_Param" name="↩
                simulationTime">
              <VALUE type="Real" value="3600"/>
            </INIT_PARAM>
            <INIT_PARAM xsi:type="Devs:T_Value_Init_Param" name="↩
                simulationRuns">
              <VALUE type="Integer" value="1"/>
            </INIT_PARAM>
          </INIT_PARAMS>
        </LIBRARY_COMPONENT>
      </COMPONENT_REFERENCE>
      <COMPONENT_REFERENCE xsi:type="Devs:T_Component_Reference"
      text="Composite-Network regional office 1 net evaluation">
        <LIBRARY_COMPONENT class="Network" package="eis">
          <INIT_PARAMS>
            <INIT_PARAM xsi:type="Devs:T_Value_Init_Param" name="↩
                throughput">
              <VALUE type="Real" value="100"/>
            </INIT_PARAM>
            <INIT_PARAM xsi:type="Devs:T_Array_Init_Param" name="nodes↩
                ">
              <INIT_PARAMS/>
            </INIT_PARAM>
            <INIT_PARAM xsi:type="Devs:T_Array_Init_Param" name="↩
                networks">
              <INIT_PARAMS>
                <INIT_PARAM xsi:type="Devs:T_Value_Init_Param" name="↩
                    network">
                  <VALUE type="String" value="Atomic-Network registry ↩
                      office 1 net evaluation"/>
                </INIT_PARAM>
                <INIT_PARAM xsi:type="Devs:T_Value_Init_Param" name="↩
                    network">
                  <VALUE type="String" value="Atomic-Network ↩
                      datatacenter 1 net evaluation"/>
                </INIT_PARAM>
              </INIT_PARAMS>
            </INIT_PARAM>
          </INIT_PARAMS>
        </LIBRARY_COMPONENT>
      </COMPONENT_REFERENCE>
      ....
  </DEVS_COUPLED>
</Devs:MODEL>
```

Listing 2. DEVS simulation model.

XML representations to Java classes, incorporating simulation results in the system model. For each element that has output parameters, these parameters are updated with values from the corresponding XML attribute.

All these tasks require expertise in several technologies and standards, as SysML, DEVS, MOF, QVT, Java and the EIS domain. However, once implemented, the benefits from their combined use are available for multiple uses by numerous interested EIS engineers.

4.5. Implementation

This subsection provides a summary of the activities performed to implement the framework and a synopsis of the NFRs verification process. First of all, attributes of the EIS profile that determine the performance of EIS components were identified. In addition, simulation execution parameters were identified in EIS evaluation scenarios. The revised DEVS meta-model, as described in Section 4.2, was used, in order to utilize existing DEVS executable components. Moreover, DEVS executable components required for EIS simulation were implemented, as no appropriate existing DEVS library components were available. For this purpose, the EIS components that should be simulated were identified, their behavior was analyzed and the

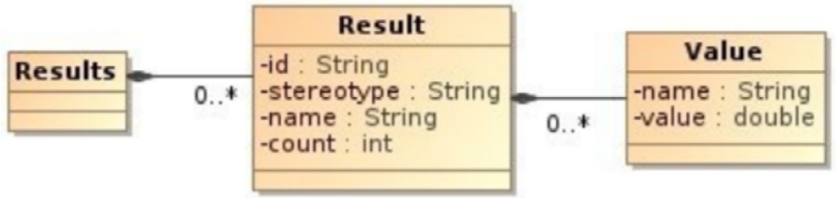

Fig. 5. Results Meta-model.

```
<results>
    <result stereotype="Eval-Atomic-Network::Eval-Service-Replica"
name="update taxation entity" id="↩
    _16_8_14d00da_1366132365894_952758_15900"
count="6">
        <value value="0.00044780904330914747" name="avg-ResponseTime"/↩
            >
        <value value="0.003437379655217227991670370101928710937 50"
name="max-ResponseTime"/>
    </result>
    <result stereotype="Eval-Atomic-Network::Eval-Service-Replica"
name="update taxation entity" id="↩
    _16_8_14d00da_1366132365900_571644_15928"
count="6">
        <value value="0.00022175926571496682" name="avg-ResponseTime"/↩
            >
        <value value="0.002102622840124240610748529434204101562 50"
name="max-ResponseTime"/>
    </result>
    ...
</results>
```

Listing 3. Sample simulation output file.

respective DEVSJava [41] components were developed. QVT relations that transform EIS models to the respective composite DEVS models, according to the extended version of the DEVS meta-model were defined. An XSLT transformation that creates Java code to instantiate and connect the appropriate DEVS executable components, as instructed by the generated DEVS model, was defined. This enabled the exploitation of the EIS DEVSJava executable components. A DEVS simulation controller that handles multiple simulation runs, receives simulation results, and exports them in a XML document, was also implemented.

Finally, to integrate simulation results in the original model, MagicDraw plugin was extended using the openAPI [46]. The diversity of processes, software and information representations remains transparent to the system designer that uses a single modeling tool.

Hence, simulation of the EIS domain using DEVS, is simplified and performed as a set of distinct steps, most of which are automated. The process is organized as indicated in the following points:

1. The system designer defines the system model in MagicDraw using the EIS profile and the EIS plugin. Different views are used for the description of different aspects of the system:
   (a) application architecture is defined in the Functional view;
   (b) system access points and topology is defined in the Topology view;
   (c) network architecture is defined in Network Infrastructure view
2. The system designer defines non functional requirements with quantitative properties;
3. Complex, non functional requirements that derive from primitive requirements are automatically calculated;
4. The system engineer creates evaluation scenarios, based on the current system model and defines general simulation parameters like simulation duration;
5. The system model, including the evaluation scenarios, is exported in XMI format;
6. The model (in XMI format) is transformed to the respective DEVS coupled model by a QVT transformation, executed with MediniQVT [47];
7. An XSLT translation generates the DEVSJava code for the DEVS simulation model;
8. Simulation is executed and the results are locally stored in XMI format and then are imported to the system model, through the MagicDraw plug-in;
9. Validation rules are applied to model elements;
10. Either the system model is verified, or some of the requirements are not met. In this case, they are clearly indicated to the system designer, so that he/she performs model re-adjustment, and the process is repeated from step 4.

5. An example from the designer's perspective

The modeling infrastructure, provided to EIS designers, was briefly described in Section 3 (EIS profile). Moreover, the approach for providing automated simulation generation capabilities from simulation-agnostic models was presented in Section 4. This mainly included the contribution provided by domain experts, simulation experts and modeling experts, during the setup phase, so that the required framework is developed.

During the design time phase, systems designers utilize the capabilities provided by the framework, for systems design and requirements verification. This practical perspective of the approach is presented in this section, using a sample EIS model.

System designers may work with EIS models, using the MagicDraw modeling tool, where the EIS profile has been imported and the corresponding plugin has been added.

As stated in Section 3, when designing a new EIS architecture, the system designer describes both software and hardware architectures in terms of Functional, Topology and Network Infrastructure views [2]. After defining the EIS architecture, the system designer evaluates it through the Evaluation view [6,32]. Evaluation scenarios defined in this view are snapshots of the predefined software and hardware architecture.

Let us consider, for the purposes of this example, an information system for a simple registry application. The registry client application interacts with software services executed in distributed database servers. Two kinds of user roles (*manager* and *staff*) might use the client application. Each role can use distinct services, while the probability of each service to be invoked is predefined, for evaluation purposes. Additionally, specific parameters of model elements must be defined to enable simulation execution. A role, connected to a specific requirement defines its behavior. In the case that the role is connected to another "behavior requirement", a different simulation scenario is made.

The system designer creates an evaluation scenario and is able to automatically construct the software and hardware evaluation diagrams (derived from other diagrams) inside the modeling tool. Then the system model is enriched with simulation-related properties, such as simulation runs, end conditions, etc, enabling its transformation to simulation model.

The scenario is consequently simulated using DEVS as proposed in Section 4. The hardware architecture diagram depicts the network architecture, where the application operates on.

Based on an evaluation scenario, the following steps are required to achieve the requirement verification process:

- Simulation configuration: For a specific scenario, the designer defines simulation parameters, such as the end condition, which is defined in terms of time, e.g. 28800s, which is 8 hours of simulation time, and the number of simulation runs. Using a popup menu, the system designer is able to run the simulation. When the simulation execution is started, intermediate steps (transformations) are executed transparently to the designer. The designer may observe execution in DEVSJava SimView environment.
- When the simulation has been completed, the designer is notified to import the results into the evaluation scenario.
- Once the results have been incorporated, verification rules are executed and the unverified model elements are annotated (see Fig. 6). Thus, the system designer is aware of model elements that should be adjusted.

Fig. 6 presents an excerpt of the evaluation view, where the simulation results have been incorporated in the system model evaluation elements. The incorporation involves an XML file, similar to the one illustrated in Listing 3, processed by the EIS plugin that supplements the values of predefined elements.

A local network consists of computers with software components running on them. For a specific network, the traffic derived from the requirements has an average value of 2.7 Mbps with a standard deviation of 0.2 Mbps, as shown in Fig. 6. The simulation results show a value of 3.1 Mbps, which is out of range. This non-verified requirement is depicted with red color in Fig. 6, since it is not satisfied by the corresponding model element. In case the system designer decides to intervene and create another simulation scenario, this would define a modified system model in correspondence with previous evaluation results.

For the application of the approach with EIS and DEVS, 9 simulation library components were developed (1200 lines of Java code), implementing the behavior of the main EIS components (network node, processing node, etc). 5 auxiliary model classes and 4 classes for handling simulation results were also developed (540 lines of Java code). The QVT transformation was defined as a set of 17 relations in a 1000 lines file. Finally, an XSLT translation (360 lines) was defined for the conversion of DEVS simulation models to executable simulation code.

The EIS model, used for the example presented in this section, has 148 classes and 832 other model elements (packages, abstractions, realizations, dependencies) and was exported from MagicDraw UML modeling tool as a 10573-lines XML document in XMI format. The EIS-to-DEVS QVT transformation was executed in a personal computer with a dual-core Intel Pentium Processor SU4100 (2M Cache, 1.30 GHz, 800 MHz FSB) and 4GB of DDR3 RAM running Ubuntu 12.04 LTS and generated the DEVS simulation model, as a 4550-line XML file in XMI format, in 38 seconds. The transformation identified the 9 network nodes, 16 servers/workstations, 58 services and 12 roles of the evaluation scenario and extracted information from other model elements in other views of the EIS model. The DEVS simulation model was converted to a simulation executable program (2457 lines of Java code) that uses the simulation library components via the XSLT translation. Simulation of 8 hours of operation of the model finished in 48 seconds, producing simulation results for 30 EIS evaluation view model entities in a 123-lines XML document, in XMI format. Simulation results were imported in the EIS model in the modeling tool were verification was completed and visualized to the designer.

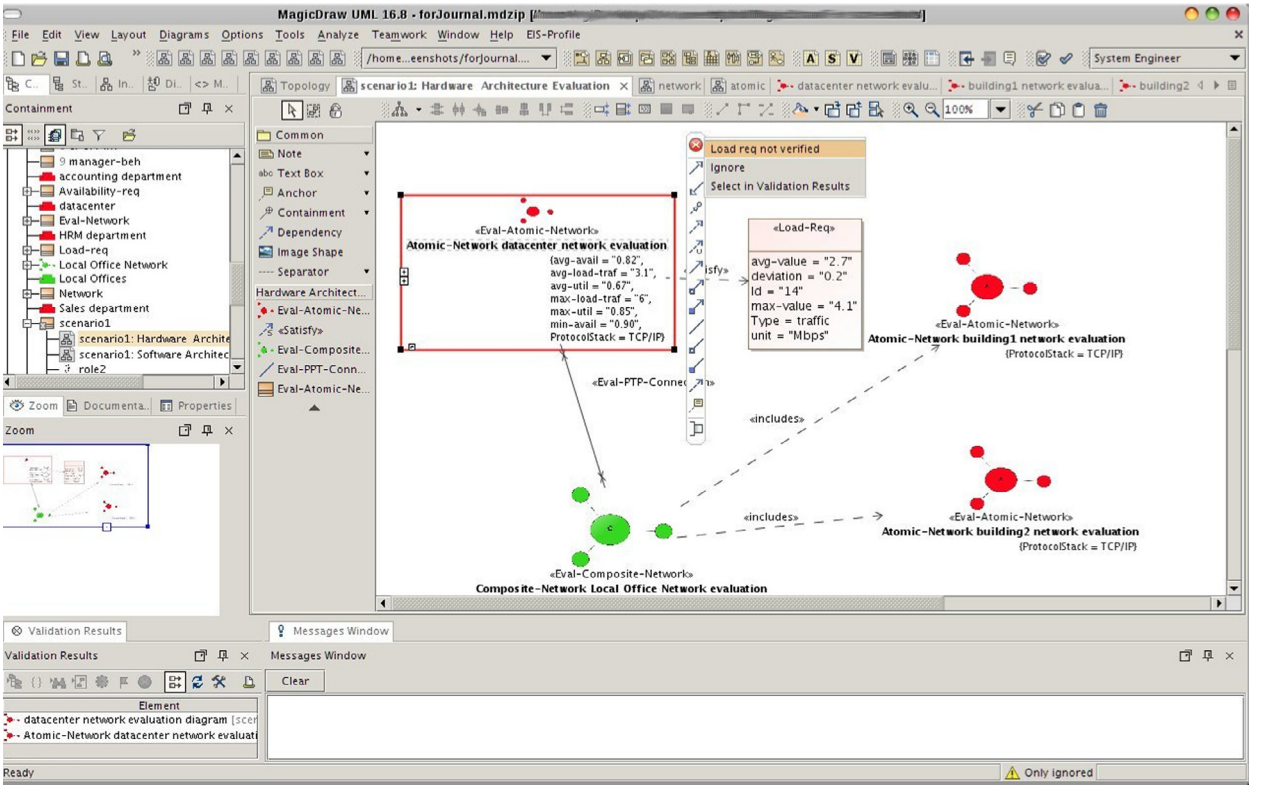

Fig. 6. Evaluation view: verifying requirements.

The benefit is that a designer is facilitated to define a system model configuration, and the modeling tool verifies NFRs of this model. The implementation needed 2100 lines of simulation code and transformations, developed only once to repetitively handle large system models (980 model elements or 10573 lines of code in this case) and produces a considerable amount of executable simulation code (2457 lines of Java code). There is no interaction between the designer and the simulation framework, thus he/she is unaware of the executable simulation models. Moreover, the designer is notified when the simulation is finished and if there are any requirements that are not satisfied.

6. Conclusions

Although SysML supports requirements modeling, SysML modeling environments do not provide facilities like automated NFRs verification. This deficiency motivated the work presented in this paper, which is an approach that enables automated, efficient verification of SysML NFRs via simulation execution and the incorporation of simulation results in the system model.

The three main axes of the approach are:

- The utilization of existing domain-specific, simulation-agnostic system models (EIS), rather than requiring refinement of system models, according to simulator-specific profiles.
- Limitation of design-time user interaction only within the system modeling tool.
- Commitment to related standards (MOF, SysML, QVT) and utilization of compliant tools.

Alignment with these axes is mainly achieved through the extension of the DEVS meta-model, so that existing simulation library components can be used, and the development of a domain-specific models to simulation-specific models transformation.

We defined specific directions and preconditions for applying the approach in the EIS domain, using DEVS as an appropriate simulation execution environment, after extending its meta-model and the meta-model-aware execution layer. SysML models are transformed to executable simulation code and the obtained simulation results are incorporated into the original SysML models through the exploitation of MDA concepts and tools. Therefore, the verification process is enhanced and simplified for the designer, as intermediate steps are automated.

We presented a set of prerequisite activities that shall be performed by domain, modeling and simulation experts, for the application of the approach on a specific domain. However, once these have been completed, all EIS designers may utilize the advanced requirements verification functionality, with limited effort.

Finally, practical aspects of the proposed requirements verification process, performed by system designers, are elaborated using a sample model. The approach and its application, denote that enforcing the role of models and staying committed to standards directs efforts originating from research and industrial communities towards solutions that enforce knowledge exchange and combined use of diverse proprietary tools. Most importantly, such an environment provides conditions that may substantially facilitate model validation and verification.

Therefore, the utilization of another simulation framework sets the next research objective. Furthermore, the vision of establishing standard simulation extensions for SysML that could be used across multiple domains and simulation frameworks becomes more robust and promising. In the direction of further enhancing the system design process, automated detection of model elements and properties that are the cause of the non-verified requirements, could assist the designer in performing system model readjustments. Such an enhancement could also include provision for automated generation of suggested readjustments.

Acknowledgment

The authors would like to thank the anonymous reviewers for their valuable comments that helped improve the presentation of the paper.

References

[1] OMG, Systems Modeling Language (SYSML) Specification, Version 1.3, 2012.
[2] A. Tsadimas, M. Nikolaidou, D. Anagnostopoulos, Handling non-functional requirements in information system architecture design, in: Proceedings of ICSEA'09, 2009, pp. 59–64.
[3] O. Batarseh, L.F. McGinnis, System modeling in SysML and system analysis in ARENA, in: Proceedings of the Winter Simulation Conference, WSC '12, 2012, pp. 258:1–258:12.
[4] G.-D. Kapos, V. Dalakas, M. Nikolaidou, D. Anagnostopoulos, An integrated framework for automated simulation of SysML models using DEVS, Simulation 90 (6) (2014) 717–744.
[5] OMG, UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems Specification, Version 1.0, 2009,
[6] A. Tsadimas, M. Nikolaidou, D. Anagnostopoulos, Evaluating software architecture in a model-based approach for enterprise information system design, in: Proceedings of SHARK '10, ACM, New York, USA, 2010, pp. 72–79. http://doi.acm.org/10.1145/1833335.1833346.
[7] H. Espinoza, D. Cancila, B. Selic, S. Gérard, Challenges in combining SysML and MARTE for model-based design of embedded systems, in: ECMDA-FA, Lecture Notes in Computer Science, vol. 5562, Springer, 2009, pp. 98–113.
[8] W. Schamai, Modelica Modeling Language (ModelicaML): A UML Profile for Modelica, Technical Report, 2009.
[9] S. Izukura, K. Yanoo, T. Osaki, H. Sakaki, D. Kimura, J. Xiang, Applying a model-based approach to IT systems development using SysML extension, in: MoDELS, Lecture Notes in Computer Science, 6981, Springer, 2011, pp. 563–577.
[10] M. Nikolaidou, G.-D. Kapos, A. Tsadimas, V. Dalakas, D. Anagnostopoulos, Simulating sysml models: overview and challenges, in: Proceedings of 2015 10th System of Systems Engineering Conference (SoSE), 2015, pp. 328–333.
[11] OMG, Model Driven Architecture. Version 1.0.1 (2003).
[12] B.P. Zeigler, H. Praehofer, T. Kim, Theory of Modeling and Simulation, 2nd ed., Academic Press, 2000.
[13] E. Huang, R. Ramamurthy, L.F. McGinnis, System and simulation modeling using SysML, in: Proceedings of the 39th Conference on Winter Simulation, WSC '07, IEEE Press, Piscataway, NJ, USA, 2007, pp. 796–803.
[14] O. Schonherr, O. Rose, First steps towards a general SysML model for discrete processes in production systems, in: Proceedings of the 2009 Winter Simulation Conference, Austin, TE, USA, 2009, pp. 1711–1718.
[15] OMG, SysML-Modelica Transformation (SyM), 2012.
[16] IBM, Rational Software Modeler (2010).
[17] NoMagic, SysML Plugin for Magic Draw (2010).
[18] A.A. Kerzhner, J.M. Jobe, C.J.J. Paredis, A formal framework for capturing knowledge to transform structural models into analysis models, J. Simul. 5 (3) (2011) 202–216.
[19] W. Schamai, P. Helle, P. Fritzson, C.J.J. Paredis, Virtual verification of system designs against system requirements, in: Proceedings of the 2010 International Conference on Models in Software Engineering, MODELS'10, Springer-Verlag, Berlin, Heidelberg, 2011, pp. 75–89.
[20] D. Knorreck, L. Apvrille, P. de Saqui-Sannes, Tepe: A sysml language for time-constrained property modeling and formal verification, SIGSOFT Softw. Eng. Notes 36 (1) (2011) 1–8, doi:10.1145/1921532.1921556.
[21] G. Behrmann, A. David, K.G. Larsen, J. Hakansson, P. Petterson, W. Yi, M. Hendriks, Uppaal 4.0, in: Proceedings of Third International Conference on Quantitative Evaluation of Systems, 2006. QEST 2006, IEEE, 2006, pp. 125–126.
[22] E. Huang, K.S. Kwon, L. McGinnis, Toward on-demand wafer fab simulation using formal structure & behavior models, in: Proceedings of Winter Simulation Conference, 2008. WSC 2008, IEEE, 2008, pp. 2341–2349.
[23] L. McGinnis, E. Huang, K.S. Kwon, V. Ustun, Ontologies and simulation: a practical approach, J. Simul. 5 (3) (2011) 190–201.
[24] O.G. Batarseh, E. Huang, L. McGinnis, Capturing simulation tool and application domain knowledge for automating simulation model creation, J. Simul. 9 (1) (2015) 1–15.
[25] S. Kruse, Co-Evolution of Metamodels and Model Transformations: An Operator-Based, Stepwise Approach for the Impact Resolution of Metamodel Evolution on Model Transformations., Books on Demand, 2015.
[26] X. Hu, Y. Sun, L. Ntaimo, Devs-fire: design and application of formal discrete event wildfire spread and suppression models, Simulation 88 (3) (2012) 259–279.
[27] A. Al-Habashna, G. Wainer, Modeling pedestrian behavior with cell-devs, Simulation 92 (2) (2016) 117–139, doi:10.1177/0037549715624146.
[28] A. Inostrosa-Psijas, V. Gil-Costa, R. Solar, M. Marin, Load balance strategies for devs approximated parallel and distributed discrete-event simulations, in: Proceedings of 2015 23rd Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), IEEE, 2015, pp. 337–340.
[29] S. Garredu, E. Vittori, J.F. Santucci, B. Poggi, A survey of model-driven approaches applied to devs - a comparative study of metamodels and transformations, in: Proceedings of 2014 International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH), 2014, pp. 179–187, doi:10.5220/0005041001790187.
[30] M. Bajaj, D. Zwemer, R. Peak, A. Phung, A. Scott, M. Wilson, Slim: collaborative model-based systems engineering workspace for next-generation complex systems, in: Proceedings of Aerospace Conference, 2011 IEEE, 2011, pp. 1–15, doi:10.1109/AERO.2011.5747539.
[31] Modelcenter, ModelCenter Integrate, http://www.phoenix-int.com/modelcenter/integrate.php.
[32] A. Tsadimas, M. Nikolaidou, 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, SAC '12, ACM, New York, NY, USA, 2012, pp. 1057–1062, doi:10.1145/2231936.2231941.

[33] P. Casas, Formal Languages for Computer Simulation: Transdisciplinary Models and Applications, Igi Global, 2013.
[34] M. Nikolaidou, G.-D. Kapos, V. Dalakas, D. Anagnostopoulos, Basic Guidelines for Simulating SysML Models: An Experience Report, in: Proceedings of Seventh International Conference on System of Systems Engineering (SoSE) 2012, 2012, pp. 95–100, doi:10.1109/SYSoSE.2012.6384172.
[35] A. Tsadimas, Model-based enterprise information system architectural design with sysml, in: Proceedings of 2015 IEEE 9th International Conference on Research Challenges in Information Science (RCIS), IEEE, 2015, pp. 492–497.
[36] M. Nikolaidou, A. Tsadimas, N. Alexopoulou, D. Anagnostopoulos, Employing Zachman enterprise architecture framework to systematically perform model-based system engineering activities, in: HICSS, 2009, pp. 1–10.
[37] A. Law, Simulation modeling and analysis, McGraw-Hill Series in Industrial Engineering and Management Science, 4th ed., McGraw-Hill, 2006.
[38] J. Cabot, R. Clarisó, E. Guerra, J. de Lara, Verification and validation of declarative model-to-model transformations through invariants, J. Syst. Softw. 83 (2) (2010) 283–302, doi:10.1016/j.jss.2009.08.012.
[39] M. Nikolaidou, A. Tsadimas, D. Anagnostopoulos, Model-based enterprise information system architecture design using SysML, in: Proceedings of IEEE Systems Conference 2010, 2010.
[40] L. McGinnis, V. Ustun, A simple example of SysML-driven simulation, in: Proceedings of the 2009 Winter Simulation Conference (WSC), IEEE, 2009, pp. 1703–1710.
[41] B.P. Zeigler, H.S. Sarjoughian, Introduction to DEVS Modeling and Simulation with JAVA, DEVSJAVA Manual, 2003.
[42] G.-D. Kapos, V. Dalakas, A. Tsadimas, M. Nikolaidou, D. Anagnostopoulos, Model-based system engineering using SysML: Deriving executable simulation models with QVT, in: Proceedings of 2014 8th Annual IEEE Systems Conference (SysCon), IEEE, 2014, pp. 531–538.
[43] Bitbucket, EIS DEVSjava Library Components, Bitbucket, 2014.
[44] J. Mather, The DEVSJAVA Simulation Viewer: A Modular GUI that Visualizes the Structure and Behavior of hierarchical DEVS Models, Ph.D. thesis, University of Arizona, 2003.
[45] Java Platform, Java Architecture for XML Binding, Standard Edition, Java Platform, 2013.
[46] NoMagic, MagicDraw Open API User Guide, 2013.
[47] IKV++ Technologies ag, Medini QVT, IKV++ Technologies ag, 2013.