Integrating Simulation Capabilities into SysML for Enterprise Information System Design

Anargyros Tsadimas, George-Dimitrios Kapos, Vassilis Dalakas, Mara Nikolaidou
and Dimosthenis Anagnostopoulos
Department of Informatics and Telematics
Harokopio University of Athens
70 El. Venizelou St, Kallithea, 17671, Athens, GREECE
email: {tsadimas, gdkapos, vdalakas, mara, dimosthe}@hua.gr

***Abstract*—Performance requirements play a significant role in the design of large-scale systems, such as enterprise information systems. Systems Modeling Language (SysML), proposed by Object Management Group (OMG) for system engineering, provides for requirements specification, though a verification method for quantitative requirements as performance ones is lacking. In the information systems domain, performance requirements are usually verified using simulation. To integrate simulation capabilities into SysML the authors have proposed the concept of the Evaluation View, a discrete diagram to specify enterprise information system architecture under evaluation and the conditions under which performance requirements should be verified. A corresponding SysML profile, called Enterprise Information System (EIS) profile, has been defined. In this paper we present an approach that provides (a) the automated transformation of SysML-based EIS models defined in the Evaluation View to executable simulation code for Discrete Event System Specification (DEVS) simulation environments and (b) the incorporation of simulation results into the original EIS SysML models to enable the verification of corresponding performance requirements.**

***Index Terms*—Model-Based System Design, SysML, Non-functional Requirements, Requirement Verification, Simulation, Model Transformations, MDA.**

I. Introduction

Enterprise Information System (EIS) design is a complex task, based on the integration of different aspects and concerns, subjected to both functional and non-functional requirements (NFRs). Especially during EIS architecture design, where the way EIS hardware and software components are interrelated is defined, NFRs, such as performance requirements, play a significant role [1], since they depict the conditions under which specific system components should operate, leading to alternative design decisions.

A model-based approach for EIS architecture design has been proposed by the authors in [2], utilizing SysML to model alternative system views, emphasizing different aspects of the system. Among them a specialized view, called NFR view, focuses on NFR description and verification extending SysML requirement diagram [3]. Emphasis is given on performance requirements that specific system components and the system as a whole should satisfy, thus the detail description and verification of quatitative requirements is provided. To serve the verification of NFRs a different view is introduced, call Evaluation View. To explore whether performance requirement are satified, in the domain of EIS simulation is usually applied, and more specifically discrete event simulation [4]. Thus, the EIS model consisting Evaluation View should be simulated, while simulation results should also be incorporated into the model to facilitate requirement verification in SysML. This is the focus of this paper.

Apparently SysML supports a variety of diagrams describing system structure and states, necessary to perform simulation, which are utilized by different approaches [5], [6]. In most cases, SysML models defined within a modeling tool are exported in 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 proposed an integrated framework to simulate SysML models using DEVS simulators [7]. A MetaObject Facility (MOF) metamodel for DEVS is proposed and used for the definition of a standards-based transformation (QVT) of enriched SysML models to DEVS models that are consequently transformed to executable DEVS code.

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 [8]. In order to generate simulation code, a SysML model should be enriched with simulation properties. In the case of EIS design, entities included in the Evaluation view should contain simulation properties to enable the simulation of EIS system models. In this paper, we focus on the ability of the proposed profile to fully automate the generation of executable simulation code for EIS models defined in SysML by the system engineer and the prospect to integrate simulation results into the system model to enhance requirement verification process.

II. Related Work

There are many efforts to simulate SysML models, utilizing both continuous and discrete event simulators. In most cases, system models are exported from SysML in XMI format and consequently fed into the simulation environment. A variaty of SysML diagrams are utilized to describe simulation models, depending on the type of simulation used and the choice of the simulation environment. Though, in all the cases a corresponding SysML profile is defined to describe simulator specific properties [8].

Simulation of discrete event systems is commonly utilized, based on system behavior described in SysML activity, sequence or state machine diagrams. In [9], 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. Model Driven Architecture (MDA) concepts are applied to export SysML models from a SysML modeling tool and, consequently, transformed into Arena simulation models, which should be enriched with behavioral characteristics before becoming executable.

The SysML4Modelica profile endorsed by the OMG [10] 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 UML extension mechanism, they can be imported in any standard UML modeling tool, such as Rational Modeler [11] or MagicDraw [12], enabling the integration of simulation tools. In [13], 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 [13]. To embed simulation capabilities within SysML, ModelicaML profile is used. Verification conditions associated to testable requirements are also defined in ModelicaML [14], while requirement verification is performed in an external modelica tool (MathModelica) through visual diagrams created during simulation. No feedback is returned within the SysML modeling tool, thus the system engineer must have notion of both the SysML modeling tool and modelica environment, while simulation results are kept seperately of corresponding system models.

As already mentioned, most of the aforementioned approaches utilize existing modeling tools to define and apply simulation-enabling profiles, while they use corresponding external simulation tools for system validation. MDA concepts are adopted to enable the model-driven transformation of SysML models to simulation code. In most cases, the simulation of models is restricted on a specific domain, as for example real-time systems in Modelica or production line system in Arena, which is supported by corresponding libraries in the specific simulation environment. System validation and requirement verification is usually left to the system designer. SysML4Modelica profile provides for requirement verification, but this is performed within Modelica tools and not using the SysML model.

In the paper, we propose that SysML requirement verfication process should be performed within the SysML modeling tool, while simulation results should be incorporated within the SysML system model and become available to the system engineer for future reference. Thus, although a specific simulation environment is used, the verification process is perforned independently from the simulation environment and the system engineer may have no notion of the simulation method or envirnoment used. In case a different simulation platform is utilized, no changes in EIS models is inflicted.

III. EIS Profile Overview

A SysML profile for EIS design has been proposed in [2] to support the basic activities identified during any system design. These are: requirement definition, solution synthesis, solution evaluation and solution re-adjustment [4]. Solution synthesis encompasses design views of EIS profile, namely *Functionality*, *Topology* and *Network Infrastructure* Views. Functionality View focuses on software architecture design, Topology View on software allocation process and Network Infrastructure View on hardware configuration and network allocation. In each of these views, the system designer explores functional requirements and architectural design [2]. A specialized view is utilized for the description of NFRs, emphazing quatitative parameters, the automated derivation of requirements based on others, as requirements imposed to entities belonging in different views may be interrelated, and verification functions [3].

The EIS profile offers the *evaluation* view [2], to serve system evaluation activity and manage evaluation results and requirement verification. In order to facilitate the requirements verification process, evaluation view encompasses entities from the aforementioned views and stores all the required attributes for each evaluated model element.

The elements that are participating in EIS profile views are associated with relations such as *satisfy*, *verify*, *allocate* and *evaluate* [7]. *Satisfy* relates system elements defined in design views with requirements, *verify* relates requirements that are verified by elements from the 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 system components defined in design views.

The evaluation view also facilitates the definition of the conditions under which the system will be evaluated the incorporation of the evaluation results and the requirement verification process, informing system designers for inconsistencies between defined requirements and system performance after the simulation process is completed. To achieve this, 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 that verify specific requirements associated with them. Regardless of the method used to perform system evaluation, these entities have input properties, related to design entities, and output properties, depicting evaluation data. Based on the value of the output properties, requirements are verified or not.

A design entity may satisfy two kinds of NFRs: performance requirements (depicting system performance restrictions) and behavior requirements (depicting system behavior). Only a performance requirement must be verified by an evaluation entity, since a behavior requirement provides input properties to the evaluation entity, indicating the conditions (constraints) under which the evaluation should be done (Fig. 1).

Each evaluation scenario consists of two sub-views (diagrams), focusing on software and hardware design, respectively. Entities of software evaluation diagram correspond to entities from functional view and define the behavior of the software components during the evaluation of the proposed EIS architecture design. Hardware evaluation diagram entities correspond to entities from topology and network infrastructure views. These are exploited in order to initialize a corresponding simulation model instance and/or evaluate the design entity and/or verify the corresponding requirements.

The EIS profile is complemented by a plug-in, developed for the MagicDraw UML tool [3]. This plug-in handles model constraints, embeds the desired functionality in MagicDraw, and supports the evaluation process.

IV. Simulating Evaluation View

A primary concern in the configuration of simulation for EIS models is the selection of an appropriate simulation framework. Simulation frameworks supporting discrete time simulation are well suited in the case of EIS, as users and software services generate requests for other services. 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 random behavior of specific parts of the system, like time handling and type of requests made by users.

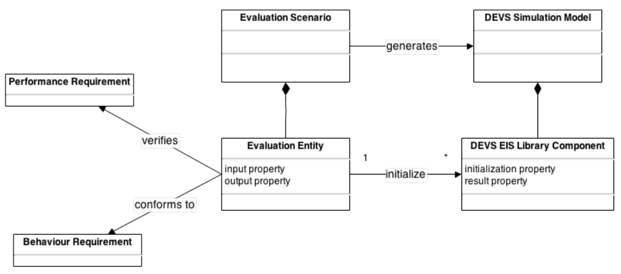


Fig. 1. Conceptual transformation

DEVS is appropriate for discrete time simulation by definition. Also, a MOF metamodel for DEVS simulation models specification is available [15], enabling the definition and execution of standards-based transformations of EIS models (UML meta-model) to simulation models. Additionally, the authors already had experience on simulating SysML models with DEVS [7]. Thus, DEVS was selected as the framework for simulation execution.

Fig. 2 illustrates the EIS simulation life-cycle, consisting of four discrete steps:

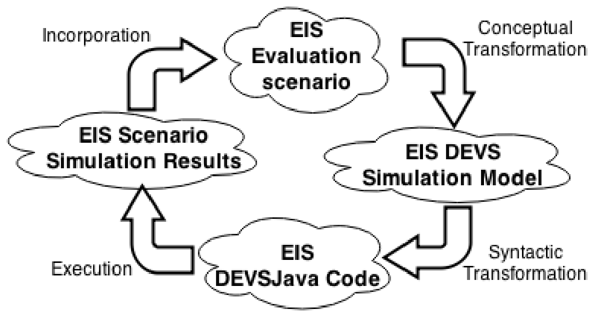


Fig. 2. EIS simulation lifecycle

- **Conceptual Transformation:** The evaluation scenario of an EIS system model is transformed to the respective DEVS simulation model. Such a transformation examines different aspects of the evaluation view of the EIS model and generates a complete, yet declarative DEVS simulation model. A powerful transformation language, like QVT is suitable in this case.
- **Syntactic Transformation:** The declarative simulation model is transformed to executable DEVSJava code. Since both simulation model and executable code share the same context (i.e. DEVS), such a transformation is mainly syntactic and is implemented with XSLT.
- **Execution:** The simulation code is executed, providing simulation results in XML format.
- **Incorporation:** Simulation results are incorporated in the EIS evaluation scenario of the system model and compared against predefined requirements.

A. Automated simulation code generation

The DEVS meta-model is considered as the input format for executable simulation models. Thus, executable simulation code generation is coarsely performed with the transformation of the EIS models to the respective DEVS models. However, as illustrated in Fig. 2, the DEVSJava environment was extended, so that it can handle DEVS models by introducing a layer that translates DEVS models to the respective DEVS Java classes.

In the case of EIS, models are composed of large amounts of interconnected components of specific types (e.g. nodes, services). Therefore, the respective, required simulation components were analyzed and implemented during the application of the approach for EIS. On the other hand, initialization, composition and interconnection of the components emerge from the EIS model during each EIS model verification execution.

Regarding the generation of DEVS models from EIS models, the structure and relationships of the latter were analyzed. The main model entities that affect overall performance were identified in both Hardware and Software Evaluation

Diagrams. Although, for EIS design purposes, it is better to define different aspects of EIS elements using different diagrams/views, the simulation model should capture a snapshot of the structure and attributes for a given concrete system. Additionally, verification and effectiveness of the simulation largely depends on the simulation model. Therefore, 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.

In a more detailed level of this transformation process, EIS entities were further analyzed to identify their key attributes that determine the performance of the system. Equivalent attributes of the respective simulation elements were defined. The transformation handles their proper initialization, based on the values of the respective attributes of EIS elements. The possible entity interconnection schemes and additional information derived from the combination with other EIS model elements were also examined.

The transformation scheme described above is abstracted in Fig. 1. An EIS evaluation scenario consists of evaluation entities that conform to behavior requirements and should verify performance requirements, as explained in III . The DEVS simulation model is generated from the evaluation scenario and consists of the respective DEVS EIS library components. The latter are initialized by the EIS evaluation entities, according to their behavior requirements. Performance requirements can be verified once simulation has been executed and simulation results are incorporated in the evaluation scenario.

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 DEVS metamodel is the specification for defining self contained DEVS models, while the EIS profile constraints ensure that no important information is missing from the EIS model. Thus, the QVT relations can successfully generate executable simulation models from any valid EIS model.

Given that the required simulation components are already implemented in DEVSJava, the essential information contained in the generated 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 (XMI) to the respective DEVSJava configuration class that instantiates and configures all EIS-related DEVSJava components, forming, this way, the executable DEVSJava code. This transformation was defined using XSLT, as it is basically a syntactic transformation that exploits initialization information in the DEVS model and creates the respective Java declarations and statements. Therefore, the generated DEVS simulation models are executed in the DEVSJava environment after an automated transformation.

B. Integrating simulation results

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 (Fig. 3).

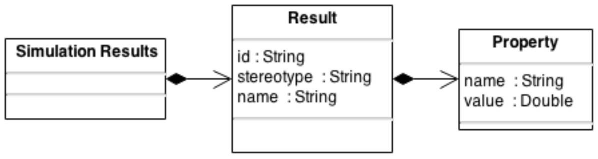


Fig. 3. Results metamodel

In the results model, for each EIS model element two properties are 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.

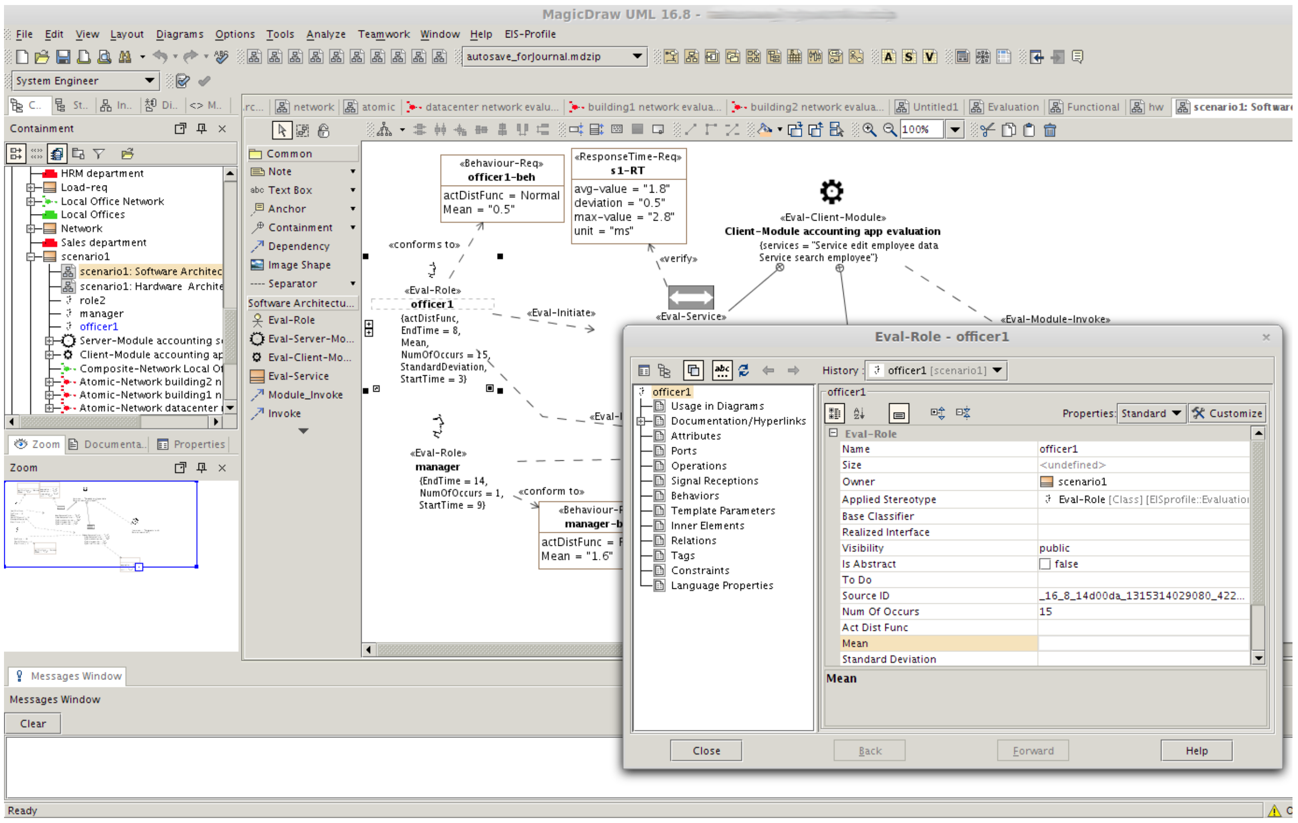
Having the simulation results imported in the system model, the only thing that the system designer should do is to run validation rules.

V. CASE STUDY

In order to be able to work with EIS models, the system designer must use the MagicDraw modeling tool. Moreover, the EIS profile and the corresponding plugin should be imported. Both software and hardware architectures are described in terms of Functional, Topology and Network Infrastructure views.

After defining the EIS architecture, the system designer is able to evaluate it through the Evaluation view [3]. Evaluation scenarios defined in this view, are snapshots of the predefined software and hardware architecture. An information system of a simple registry application is considered in Fig. 4. 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 depicted in Fig. 4 is connected to a specific requirement that defines its behavior.

System designers create evaluation scenarios and are able to automatically construct the software and hardware evaluation diagrams (derived from other diagrams) inside the modeling tool. Then, the system models are enriched with simulation-related properties, enabling their transformation to simula-



Eval-Role - officer1

History: officer1 [scenario1]

officer1
- Usage in Diagrams
- Documentation/Hyperlinks
- Attributes
- Ports
- Operations
- Signal Receptions
- Behaviors
- Template Parameters
- Inner Elements
- Relations
- Tags
- Constraints
- Language Properties

Properties: Standard · Customize

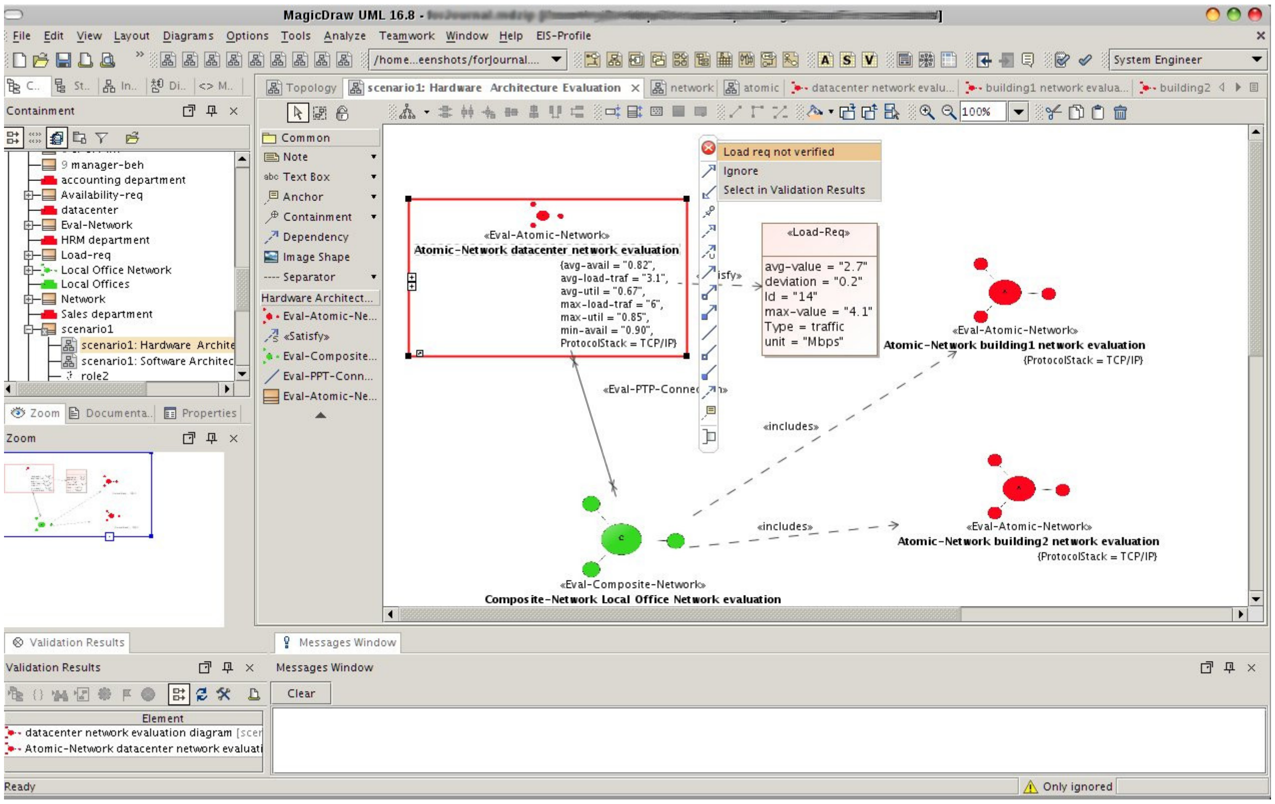
Eval-Role

Name	officer1
Size	<undefined>
Owner	scenario1
Applied Stereotype	Eval-Role [Class] [EISprofile::Evaluatio
Base Classifier	
Realized Interface	
Visibility	public
Is Abstract	false
To Do	
Source ID	_16_8_14d00da_1315314029080_422...
Num Of Occurs	15
Act Dist Func	
Mean	
Standard Deviation	

Mean

Close · Back · Forward · Help

Fig. 4. Evaluation view: Software architecture diagram



Load req not verified
Ignore
Select in Validation Results

Element
- datacenter network evaluation diagram [scer
- Atomic-Network datacenter network evaluati

Fig. 5. Evaluation view: Verifying requirements

tion models. The scenarios are consequently simulated using DEVS.

Fig. 5 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 1, processed by the EIS plugin that supplements the values of predefined elements.

```
<simulation_results>
  <result id="_16_8_14d00da_1366132365894_952758_15900"
      stereotype="Eval-Atomic-Network::Eval-Network"
      name="Atomic-Network datacenter network evaluation">
    <property name="avg-avail"
                 value="0.82"/>
    <property name="avg-load-traf"
                 value="3.10"/>
    <property name="avg-util"
                 value="0.67"/>
    <property name="max-load-traf"
                 value="6.00"/>
    <property name="max-util"
                 value="0.85"/>
    <property name="min-avail"
                 value="0.90"/>
  </result>
   ...
</simulation_results>
```

Listing 1. Sample simulation output file

Suppose we have a local network where the traffic that is derived from the requirements of the processes that are running over computers in this network, has an average value of 2.7 Mbps with a deviation of 0.2Mbps. After running the simulation, the results show a value of 3.1 Mbps, which is out of defined range between [2.5 , 2.9] Mbps. To this end, the designer could either relax the non-satisfied requirement by increasing the range or by changing the allocation of services in order to reduce the network traffic. The latter is depicted with red color in Fig. 5, where this requirement is not satisfied by the model element. 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, by clicking on them, to identify the non-verified requirement. In case the system designer decides to intervene and modify design views in line with previous evaluation results, he/she may consequently create another evaluation scenario to be simulated. Previous simulation scenarios and results are kept as part of pre-existing evaluation scenarios, enabling the system designer to keep track of the modifications made and the corresponding results, as evaluation scenarios are maintained as parts of the Evaluation View. It is up to the system designer whether an evaluation scenario will be kept or deleted.

VI. Conclusions

Verifying performance requirements is important during EIS architecture design. To this end, in this paper we presented how to integrate simulation capabilities within the EIS SysML profile, targeting model-based design of EIS architectures. Based on system models defined in SysML, corresponding simulation code is automatically generated for DEVS simulators. The tasks performed during simulation configuration require expertise in several technologies and standards, as MOF, SysML, DEVS, QVT, Java and the EIS domain. However, once the simulation libraries and corresponding transformations were implemented for the EIS domain, the benefits from their combined use become available for multiple uses by EIS engineers using SysML to explore enterprise information system design scenarios. Based on our experience on EIS domain, it is evident that a complex issue, as SysML model simulation and moreover the incorporation of simulation results within the SysML model in an automated manner is feasible.

Future work will focus on applying the proposed simulation approach for SysML models in other domains, as well as the incorporation of alternative simulation environments.

References

[1] 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.

[2] 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.

[3] ——, "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. [Online]. Available: http://doi.acm.org/10.1145/2231936.2231941

[4] J. A. Estefan, *Survey of Model-based Systems Engineering (MBSE) Methodologies - Revision B*, INCOSE MBSE Focus Group, June 2008.

[5] E. Huang, R. Ramamurthy, and L. F. McGinnis, "System and simulation modeling using SysML," in *WSC '07: Proceedings of the 39th conference on Winter simulation*. Piscataway, NJ, USA: IEEE Press, 2007, pp. 796–803.

[6] O. Schonherr and 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, December 2009, pp. 1711–1718.

[7] P. Casas, *Formal Languages for Computer Simulation: Transdisciplinary Models and Applications*. Igi Global, 2013. [Online]. Available: http://books.google.gr/books?id=YDismgEACAAJ

[8] M. Nikolaidou, G.-D. Kapos, V. Dalakas, and D. Anagnostopoulos, "Basic Guidelines for Simulating SysML Models: An Experience Report," in *Proc. Seventh Int. Conf. on System of Systems Engineering (SoSE) 2012*, July 2012, pp. 95–100.

[9] O. Batarseh and L. F. McGinnis, "System modeling in sysml and system analysis in arena," in *Proceedings of the Winter Simulation Conference*, ser. WSC '12. Winter Simulation Conference, 2012, pp. 258:1–258:12. [Online]. Available: http://dl.acm.org/citation.cfm?id=2429759.2430107

[10] OMG, *SysML-Modelica Transformation (SyM)*, Nov. 2012. [Online]. Available: http://www.omg.org/spec/SyM/1.0/PDF/

[11] IBM, "Rational Software Modeler," *Available online via http://www.ibm.com/developerworks/rational*, 2010.

[12] MG, *SysML Plugin for Magic Draw*, 2007.

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

[14] 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. [Online]. Available: http://dl.acm.org/citation.cfm?id=2008503.2008514

[15] G.-D. Kapos, V. Dalakas, A. Tsadimas, M. Nikolaidou, and D. Anagnostopoulos, "Model-based System Engineering using SysML: Deriving Executable Simulation Models with QVT," in *Systems Conference (SysCon), 2014 IEEE International*. IEEE Computer Society, 2014.