Traceability Links in Model Transformations between Software and Performance Models

Mohammad Alhaj and Dorina C. Petriu

Dept. of Systems and Computer Engineering, Carleton University,
1125 Colonel By Drive, Ottawa, Ontario, Canada, K1S 5B6
{malhaj,petriu}sce.carleton.ca

Abstract. In Model Driven Engineering, traceability is used to establish relationships between various software artifacts during the software life cycle. Traceability can be also used to define dependencies between related elements in different models, to propagate and verify properties from one model to another and to analyze the impact of changes. In this paper we describe how to define typed trace-links between different kinds of models in our model transformation chain PUMA4SOA, which generates Layered Queuing performance models from UML software models of service-oriented applications. The goal of PUMA4SOA is to help evaluate the performance of SOA systems in the early development phases. In our approach, the traceability links are stored externally in a new model, which maintain traces separately from the source and target models they refer to. We illustrate how traceability links can be used to propagate the results of the performance model back to the original software model.

Keywords: Software Performance Engineering, SOA, Traceability, Trace-Links, Aspect-oriented modeling, Model transformation, Performance Analysis.

1 Introduction

Model-Driven Engineering (MDE) is a software development paradigm that changes the focus from code to models. Many models of different types are used to describe the software under development in different lifecycle phases and at different levels of abstractions. Models in different modeling languages are created, updated and transformed either manually or automatically. This raises challenges related to the ability of managing and configuring the software models. In order to improve the coherence, and consistency of models used in an MDE process, it is useful to establish and maintain trace-links between models. Traceability is a known software approach used to establish relationships between various software artifacts (including all kinds of models) during the software life cycle. This allows the developers to understand the relationships and dependencies between artifacts, to maintain their consistency and to analyze the impact of changes in different artifacts.

F. Khendek et al. (Eds.): SDL 2013, LNCS 7916, pp. 203–221, 2013.

A wide range of traceability approaches have been discussed in the literature. The survey in [1] discusses the state of the art of traceability approaches in MDE, classifying them into three categories: 1) *requirement-driven*, 2) *modeling*, and 3) *transformation* approaches. In requirement-driven approaches, traceability is defined in the requirement models as "the ability to describe and trace the requirement specifications forward and backward in the life cycle during the software development" [2]. The modeling approaches are focusing on using metamodels and models to define trace-links. In the transformation approaches, the traceability details are generated by using model transformations. This can be done by creating trace-links between the source and target model elements during the model transformation. In terms of storing and managing traceability, two approaches are proposed in [3]: the *intra-model* and the *extra-model* approach. In the intra-model approach, traceability links are embedded inside the models they refer to as new model elements. In the extra-model approach, traceability links are stored externally in a new model, to maintain traces separately from the model they refer to. In terms of capturing the trace-links, [4] proposes two categories: *explicit trace-links* captured directly in the models using a suitable concrete syntax (such as UML dependencies), and *implicit trace-links* generated by a model operation (such as transformation or comparison).

Performance from Unified Model Analysis for SOA (PUMA4SOA) is a modeling framework introduced by the authors in [5,6] and [7], which generates a Layered Queueing Network (LQN) model from the UML design model of a SOA system; the LQN model is then used for analyzing the performance characteristics of the SOA system in early phases of the software life cycle. PUMA4SOA extends the PUMA framework developed previously in our research group [8], as presented in the next section. PUMA4SOA in its present state does not provide trace-links between the elements of the source and target models, which are needed for tracing, analyzing or propagating the impact of changes between different models.

The focus of this paper is on defining a traceability model for PUMA4SOA, which establishes trace-links between the elements of its different models. The paper is organized as follows: Section 2 gives a high-level view of the transformation chain in PUMA4SOA. Section 3 presents the proposed traceability metamodel, which defines trace-links between UML, CSM (Core Scenario Model) and LQN model elements; the metamodel is also extended to handle cases where aspect models are used. Section 4 illustrates the use of the traceability model with a Purchase Order system example. Section 5 presents related work and Section 6 gives the conclusion and directions for future work.

2 PUMA4SOA Transformation Principles

The PUMA4SOA transformation chain is described in Fig. 1. It takes as input three UML design models: 1) platform independent model (PIM), 2) deployment diagram, and 3) aspect platform models. The SOA systems are modeled

in UML [9] extended with two OMG standard profiles: MARTE [10] for adding performance annotations and SoaML [11] for describing the service architecture.

After getting the UML input design and selecting the generic aspect models for the platform operations required in the model, the next step is to transform the UML PIM model and the aspect models into intermediate models called Core Scenario Models (CSM [12]. The purpose of CSM is to bridge the semantic gap between the UML input design model and various performance models that could be generated. At the CSM level, the aspect platform models are composed with the platform independent model to generate the platform specific model (PSM). Transforming the CSM PSM to LQN is the final step in the model transformation chain [6,8]. The LQN model is an extension of queuing networks with the capability of representing nested services [13]. An LQN model defines a set of tasks representing software processes (threads) or hardware devices, which offer services called entries. An entry of a task can make a request to an entry of another task. Once the LQN model is generated, an existing LQN solver is used to produce the performance results (such as response time, throughput, and utilizations). The results are then fed back to the UML input design for further analysis.

The UML platform independent model (PIM) describes the structural and behavioral views of SOA systems at three levels of abstractions: a) the workflow model representing the business process layer, b) the service architecture model describing the invoked services, the participants, ports and service contracts, and c) the service behavior model giving details about the behavior of the invoked services. The deployment diagram represents the configuration of the SOA system, showing the allocation of software to hardware resources. The aspect platform models represent platform operations provided by the underlying service middleware, such as service invocation, service publishing and service discovery. Each aspect model can be seen as a template with generic parameters, which will be bound eventually to concrete values just before the respective aspect will be composed with the PIM in all places (join points) where a platform operation needs to be executed. For instance, a "service invocation" aspect will be composed with the PIM for every service invocation contained in the PIM. The aspect composition can take place at three levels, as discussed in [7]: UML, CSM or LQN level. In Fig. 1, the aspect composition is performed at the CSM level, which has certain advantages [7]. The final result of the composition is a platform specific model (PSM) expressed in this case in CSM.

PUMA4SOA also defines a so-called *performance completion (PC) feature model* that represents the variability (i.e., alternatives) in the service platform. It provides the choice to select between multiple aspects based on the business requirements for the given application. The "performance completion" concept was introduced in [14], where "completions" close the gap between the abstract design models and the functions provided by a platform external to the design model. In [15], a PC feature model is used to define the variability in platform choices, execution environments and other external factors that might impact the system performance. A concrete example of PC-feature model can be found in [7].

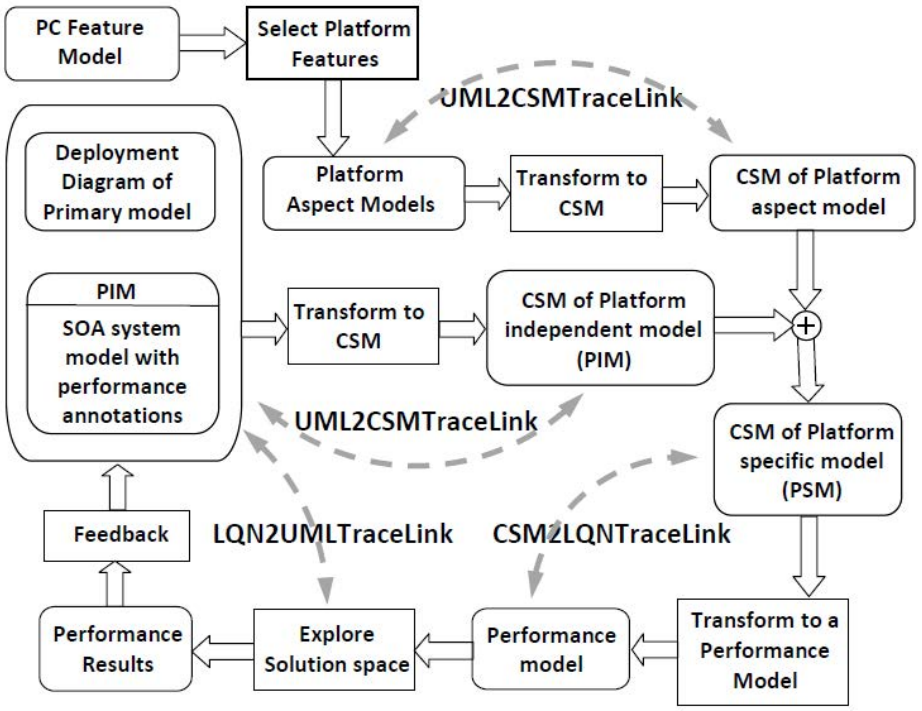

Fig. 1. PUMA4SOA approach

A high-level view of the transformation chain from UML software models to CSM and then to LQN is shown in Fig. 2. Please note that only the behavioural view of the UML model is presented in Fig. 2; the service architecture, the deployment and the platform aspect models are not shown, although they do contribute to the CSM derivation. Figure 2 emphasizes the fact that workflow and the service-providing components are represented separately in the UML model, contributing to distinct parts of the CSM and LQN models. The basic transformation principles are as follows:

a) A UML workflow model (normally an activity diagram) will generate a top-level scenario in CSM, which in turn will generate a LQN reference task that embeds an LQN activity graph corresponding to the workflow activities.
b) A service-providing run-time component in UML will generate a component in CSM executing subscenarios that represent the behaviour of services; each service subscenario is invoked by the workflow steps or other services. In turn, this will generate an LQN task with entries modeling each provided service.
c) An activity or an execution occurrence corresponding to a message in UML will generate a Step in CSM and an activity or phase in LQN.

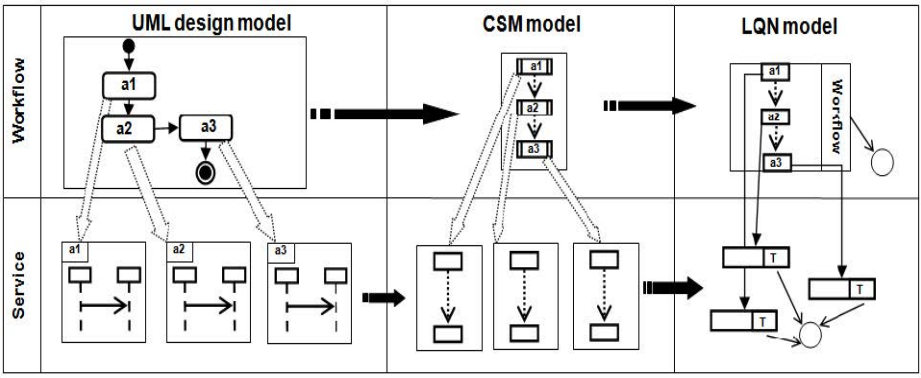

Fig. 2. Transformations in PUMA4SOA

d) A processing node in UML will generate a processor in CSM and a hardware device representing a processor in LQN.
e) An aspect model representing a platform operation will generate a subscenario in CSM, which will be woven into the PIM model when such operations are called.

3 Traceability Metamodel of PUMA4SOA

We used an approach similar to [16] for defining the PUMA4SOA traceability metamodel. The trace-links are classified into three groups: *UML2CSMTraceLink* between the UML and CSM elements, *CSM2LQNTraceLink* between CSM and LQN elements and *LQN2CSMTraceLink* between LQN and UML elements. The first two correspond to the model transformations shown in Fig. 2, while the third can be derived from the first two and it is used for feeding back to the UML model the LQN results. The three trace-links groups are aggregated into *TraceModel* (see Fig. 3).

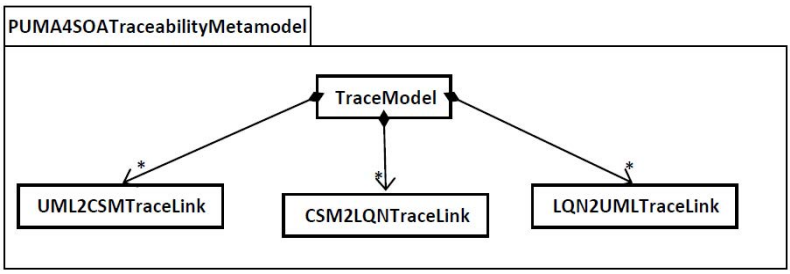

Fig. 3. PUMA4SOATraceabilityMetamodel top level

3.1 UML to CSM Traceability

The first group of trace-links in PUMA4SOA is defined between the elements of the UML input design models and the CSM model. The UML models are built using several types of UML diagrams, i.e. activity diagram (AD), component diagram, sequence diagram (SD) and deployment diagram. For simplicity, we will show here the trace-links for a subset of UML model elements.

To define *UML2CSMTraceLink*, we use the UML metamodel and the CSM metamodel, as the purpose is to capture trace-links between models elements which conform to those two metamodels. Each trace-link between an element of the source model and an element of the target model has its own type. It also has two associated properties (modeled as association roles in UML): the *source* refers to a metaclass in the *UMLMetamodel* and the *target* to a metaclass in the *CSMMetamodel*. An example of trace-link type between an *Action* element in the UML metamodel and a *StepType* element in the CSM metamodel is *Action-StepTypeTL*. The trace-links are derived during model transformation from to the mapping between corresponding UML and CSM elements. The relationships between the source and target model elements can be one-to-one (such as *Node* and *ProcessingResource*), one-to-many, many-to-one (such as *ActivityPartition*, *LifeLine* and *ComponentType*), or many-to-many.

All trace-link metaclasses inherit from *UML2CSMTraceLink*, which is aggregated into *TraceModel*. Figure 4 presents a subset of the traceability metamodel between UML and CSM. A subset of UML and CSM metamodels are represented at the top and the bottom of the figure, respectively.

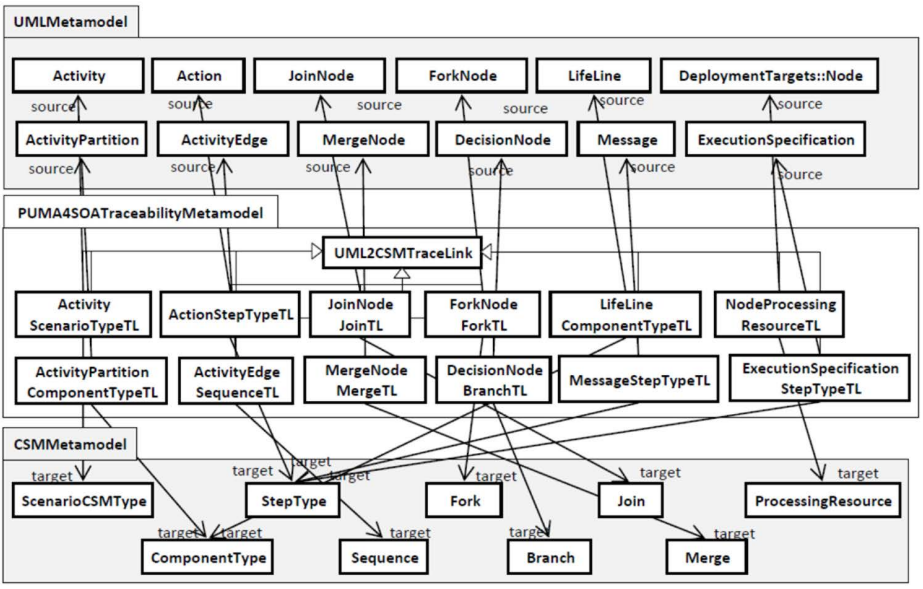

Fig. 4. Trace-Links between the elements of UML and CSM

3.2 CSM to LQN Traceability

The next group of trace-links in PUMA4SOA is defined between the elements of CSM and LQN model. We use the same procedure as in the previous section. The purpose is to capture the traceability between models that conform to the CSM metamodel and to the LQN metamodel. The metaclasses in PUMA4SOA *TraceabilityMetamodel* have two associated properties: the *source* refers to a metaclass in the *CSMMetamodel* and the *target* to a metaclass in the *LQN-Metamodel*. When an element in the source or the target model is not mapped during the model transformation, it means that it does not have equivalent element(s) in the other model. In this case a trace-link will not be defined for this element. As an example, the *OutputResultType*, which is defined in the LQN metamodel to create elements that store the results, is not linked with a CSM element; however it will have trace-links to an UML element to propagate the LQN output results. Figure 5 presents a sample of trace-links between the CSM and LQN.

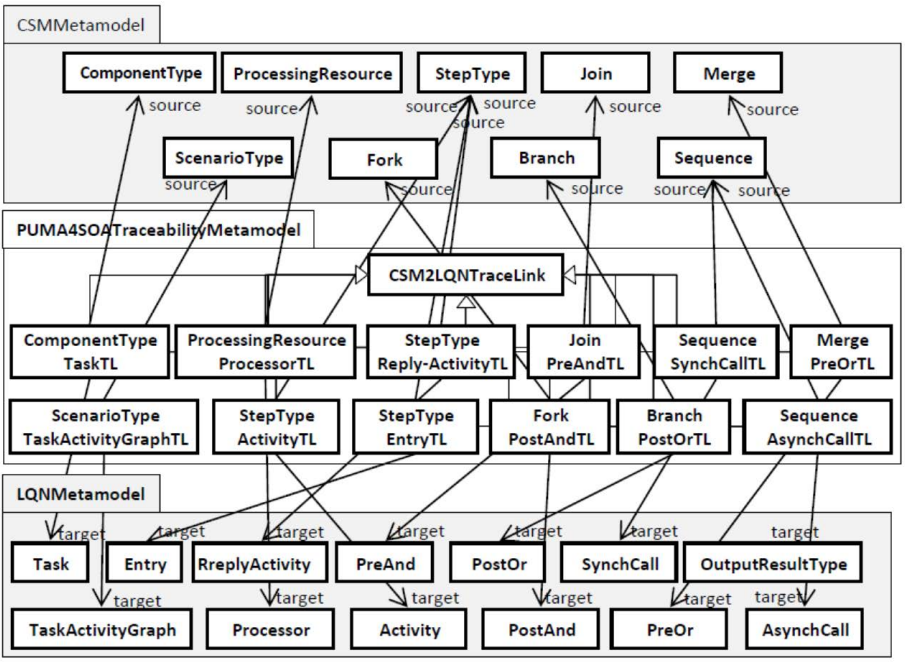

Fig. 5. Trace-Links between the elements of CSM and LQN

3.3 LQN to UML Traceability

The third group of trace-links in PUMA4SOA is defined between the LQN and UML model elements, and can be derived from the combined effect of the two

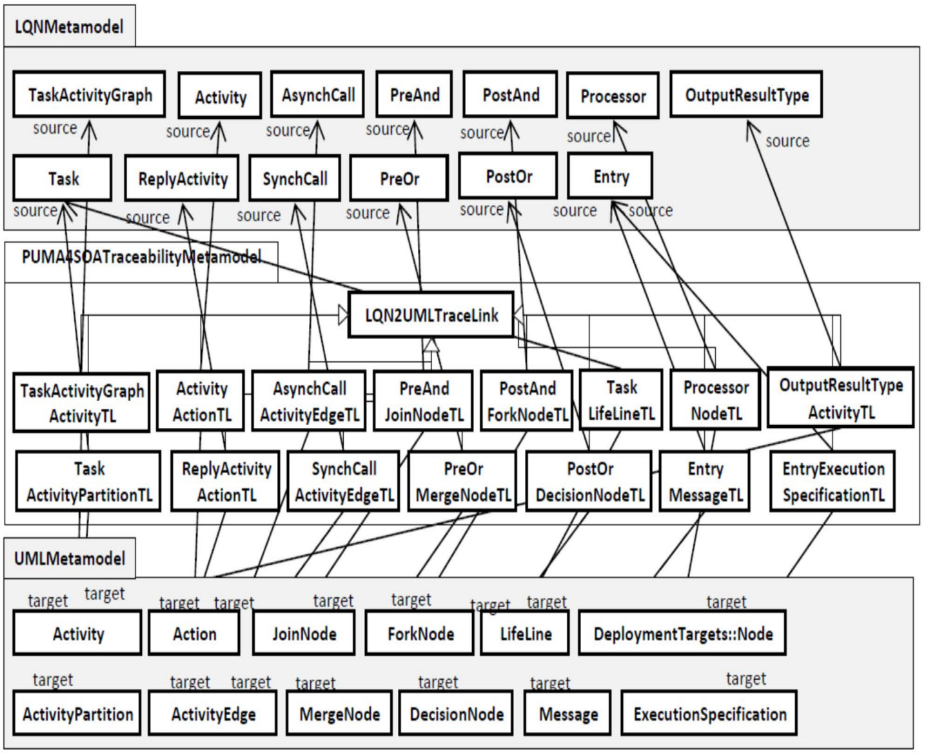

Fig. 6. Trace-Links between the elements of LQN and UML

transformations from Fig. 2, as there is no direct transformation from UML to LQN. The LQN2UML metaclasses define two associated attributes: the *source* refers to a meta-class in the *LQNMetamodel* and the *target* to metaclass in the *UMLMetamodel*. Figure 6 presents a sample of trace-links between LQN and UML.

As shown in Fig. 1, the LQN model is derived by a model transformation from the platform dependent CSM model, which in turn was generated by composing the platform aspect models into the PIM at the CSM level. Since the UML model does not contain a PSM, the mapping from LQN to UML encounters some difficulties, as discussed in the next section.

3.4 Trace-Links Related to Aspect Models

The trace-links between the LQN and UML models have not been properly defined yet in Section 3.3, because the LQN model is a Platform Specific Model (PSM), while the UML input design models do not contains a PSM, only the PIM and Generic Aspect Models (see Fig. 7). For instance, in this example we modeled the service invocation operation as a generic aspect, which involves

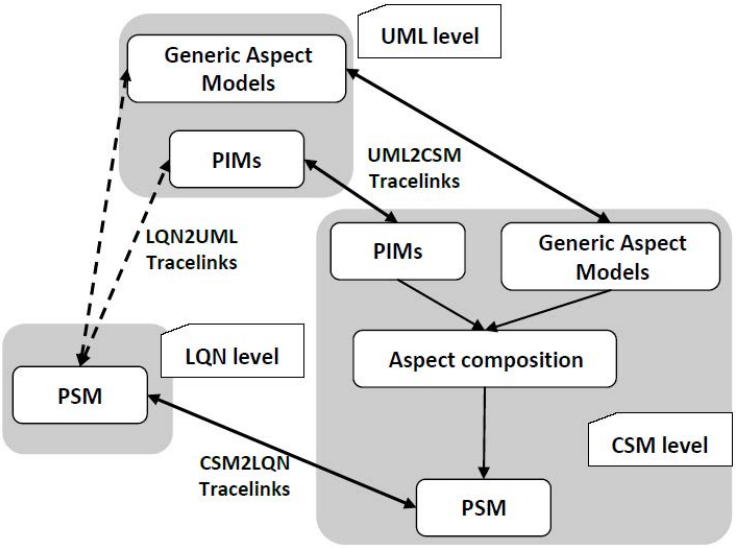

Fig. 7. Traceability for aspect models

XML parsing and marshaling/unmarshaling of the SOAP messages exchanged between the service client and the service provider. These actions are usually performed by a service middleware process, which may be instantiated multiple times on different processors, depending on the deployment of the components providing and requesting services.

Some of the concrete LQN elements obtained from the instantiation of a generic aspect model (such as the concrete instance of a service middleware used for a certain service invocation) cannot be traced back to a UML element, since only the generic middleware counterpart exists in the UML model. In this section, we extend the traceability metamodel defined in the previous sections to address this issue. The extension is used to define additional trace-links between some of the already defined trace-links. These extended trace-links allow for the mapping of the generic elements from the CSM level to LQN level. If LQN carries information about the generic model element corresponding to each concrete model element, trace-links between the concrete LQN elements can use that information to point to the generic UML counterpart.

Figure 8 shows an example of how the PUMA4SOA *TracibilityMetamodel* was extended. Two extra trace-links are created that make it possible to link a concrete LQN element (such as a concrete middleware task) with information about its generic counterpart (the generic role representing the middleware process in the aspect model). The first trace-link *ProcessorProcessorTL* defines the trace-links between *NodeProcessingResourceTL* (UML2CSM) and *ProcessingResourceProcessorTL* (CSM2LQN), and it owns two associated attributes: the *concreteSource* to define the concrete LQN *Processor*, and the *genericTarget* to define the generic CSM *ProcessingResource*. The function *SetProcessor* is

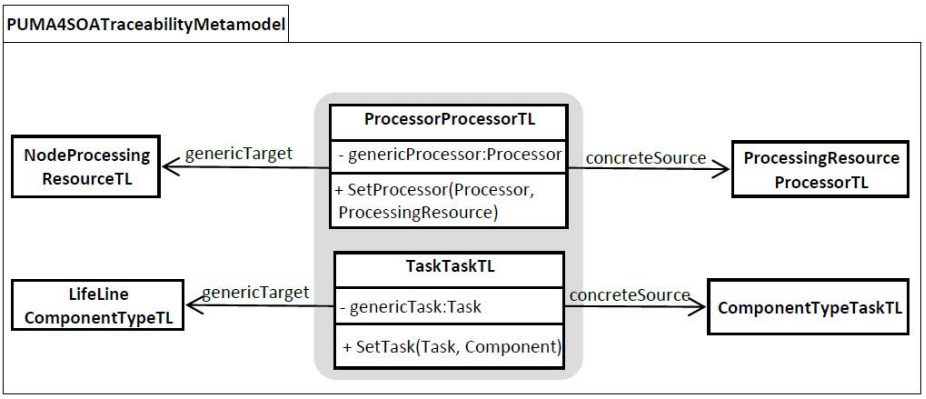

Fig. 8. Extension of PUMA4SOATraceabilityMetamodel

used to set an instance of the defined attribute *genericProcessor* to its equivalent generic CSM *ProcessingResource*. The second *TaskTaskTL* defines the trace-links between *LifeLineComponentTypeTL* (UML2CSM) and *ComponentTypeTaskTL* (CSM2LQN), and it also owns two associated attributes: the *concreteSource* to define the concrete LQN *Task*, and the *genericTarget* to define the generic CSM *ComponentType*. The function *SetTask* is used to set an instance of the defined attribute *genericTask* to its equivalent generic CSM *ComponentType*. By using such trace-links, a concrete middleware task running on a concrete processor in the LQN model can be traced to the generic UML counterparts and can also indicate the context in which they were instantiated.

4 Example: Traceability Model of Purchase Order System

In previous work [7], we used PUMA4SOA to build the UML design model of a Purchase Order (PO) system and to generate a LQN model in order to study its performance properties. In this section, we use the same example to create the traceability model which defines the trace-links between the model elements at UML, CSM and LQN levels.

A brief description of the PO case study is given first. The platform independent model (PIM) contains two parts: a) the workflow (Fig. 9) represented as a UML activity diagram, which describes the actions of receiving, invoicing, scheduling and shipping an order; and b) the service behavior models, which describe the details of each activity in the workflow.

ProcessSchedule (Fig. 10) is an example of service behavior model; the detailed models for the rest of the activities from Fig. 9 are similar, but not shown.

The deployment diagram (Fig. 11) shows the allocation of software components to the hardware nodes. The aspect platform models describe the structure and behavior of the platform operations (in this case the service middleware used

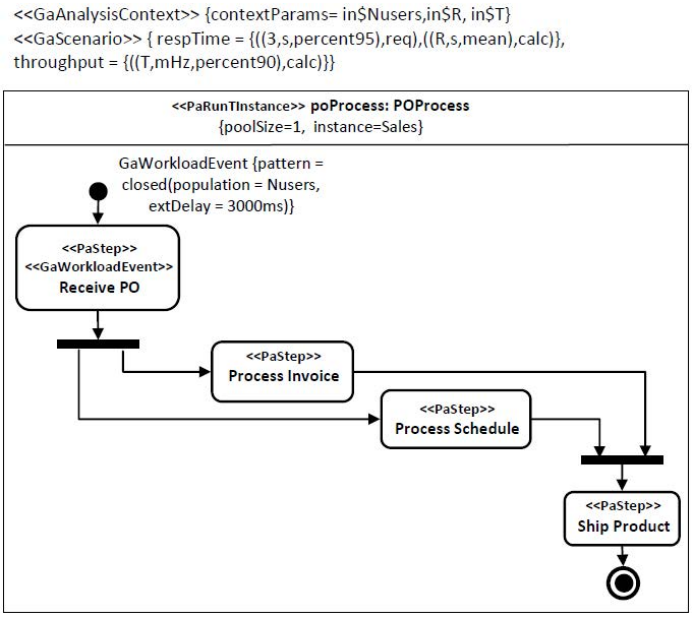

Fig. 9. Workflow model of Purchase Order system

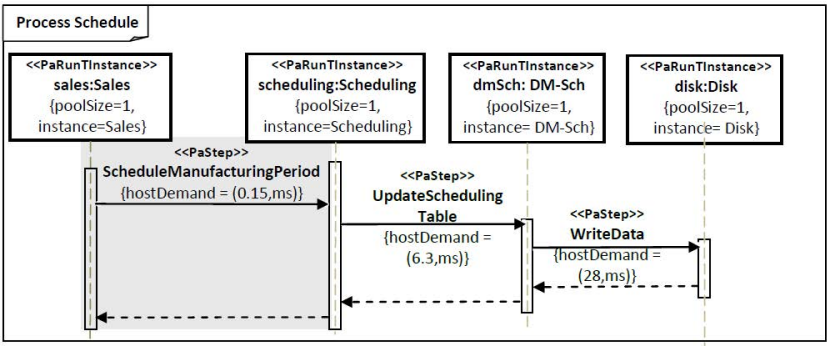

Fig. 10. Service behavior model of *ProcessSchedule*

for service invocation, discovery, publishing) in a generic format. Each middleware operation is represented by a different aspect model that has a structural and behavioural view. In this example we use only the Service Invocation operation, which describes the message construction (including XML parsing and marshaling/unmarshaling) and sending/receiving of the SOAP messages for service request and service reply, respectively (for more details, please refer to [7]).

At the CSM level, two separate CSM models are generated: one from the UML PIM (Fig. 12) and the other from the UML generic platform aspect model (service invocation in our case). Figure 12 contains two CSM scenario graphs

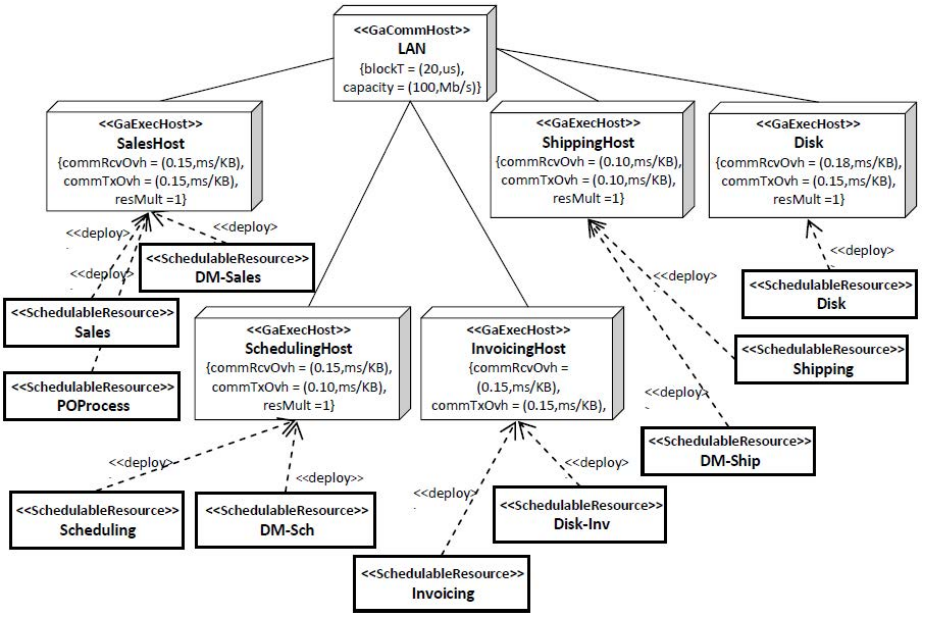

Fig. 11. Deployment diagram of Purchase Order system

composed of steps: the left one represents the workflow CSM corresponding to Fig. 9, and the right one the details of the composite step *ProcessSchedule* obtained from Fig. 10. The Aspect Oriented Modeling (AOM) technique is then used to generate a platform specific model (PSM) in CSM by composing the instantiated aspect models into the primary model (Fig. 13). The gray steps from Fig. 13 represent the woven aspects for the service request and reply.

The LQN model (Fig. 14) is generated next from the CSM Platform Specific Model. After generating the LQN model, the LQN solver produces complete performance results for the PO system, such as utilization of all resources, response times and throughput for scenarios, etc. By identifying the performance hotspots in the system, the results are fed back to the UML level to improve the software design models. Trace-links are used to propagate the performance results and to feed back the suggested improvements from the LQN to UML.

The workflow model has two MARTE stereotypes in Fig. 9. The first stereotype *<<GaAnalysisContext>>* defines the *contextParams* attribute which declares two variables: *$Nusers* for number of users and *$R* for response time. The second stereotype *<<GaScenario>>* captures system-level behavior and attaches allocations and resource usage to it. It defines many attributes, such as *respTime*, *utilization* and *throughput*. In our case, the *respTime* has two values: a required value (no more than 3 seconds) and a calculated value that will be assigned to *$R*, (the variable defined in the *contextParams*). The *$Nusers*

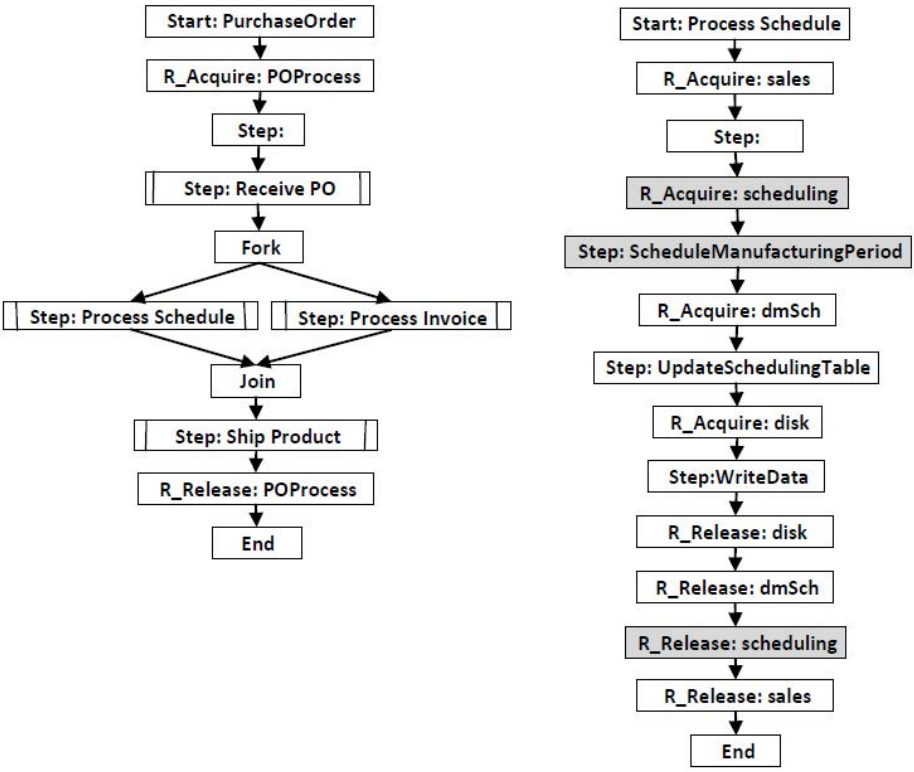

Fig. 12. The CSM model of PIM for Purchase Order system

variable is initialized by the modeler, and the $R variable will be assigned output results from the LQN model using trace-links.

PUMA4SOA allows for different types of performance analysis, such as sensitivity analysis, pass/fail and finding optimal values, which may require multiple iterations of the model transformation chain. In our case, the performance requirement specifies that the response time should be maximum 3s in average, which may require multiple model changes to achieve it. Hence, to manage the propagation of the model changes, we define a traceability model as in Fig. 15. For simplicity, trace-links for only three UML elements are defined: *POSystem:Activity*, *Sales:LifeLine* and *InvoicingHost:Node*. Some of the model elements have more trace-link relationships. For example, there are two trace-links defined between some LQN elements and the UML element *POSystem:Activity*. The trace-link between *POSystem:TaskActivityGraph* and *POSystem:Activity* corresponds to the relationship caused by the model transformation that generated LQN from UML. The other trace-link between *results:OutputResultType* and *POSystem:Activity* is used to propagate the generated output result of the LQN, such as response time and throughput. The *OutputResultType* is defined

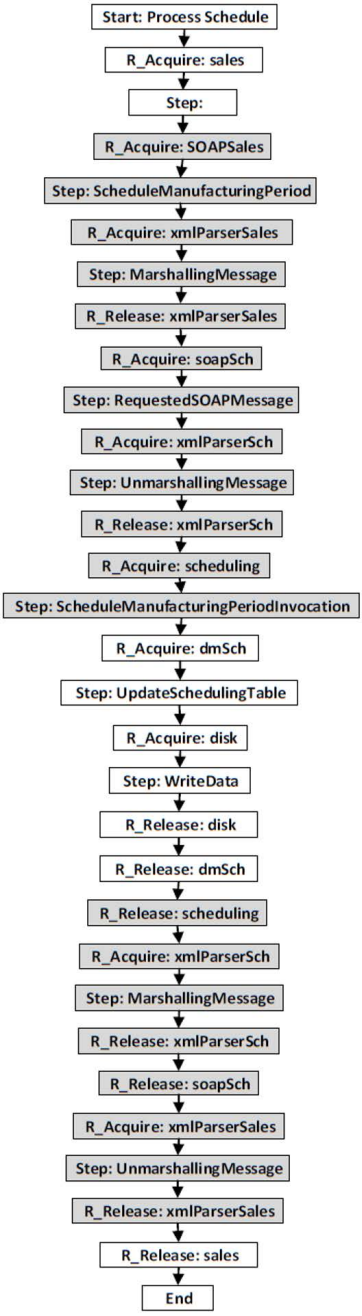

Fig. 13. Result of composition for *ProcessSchedule*

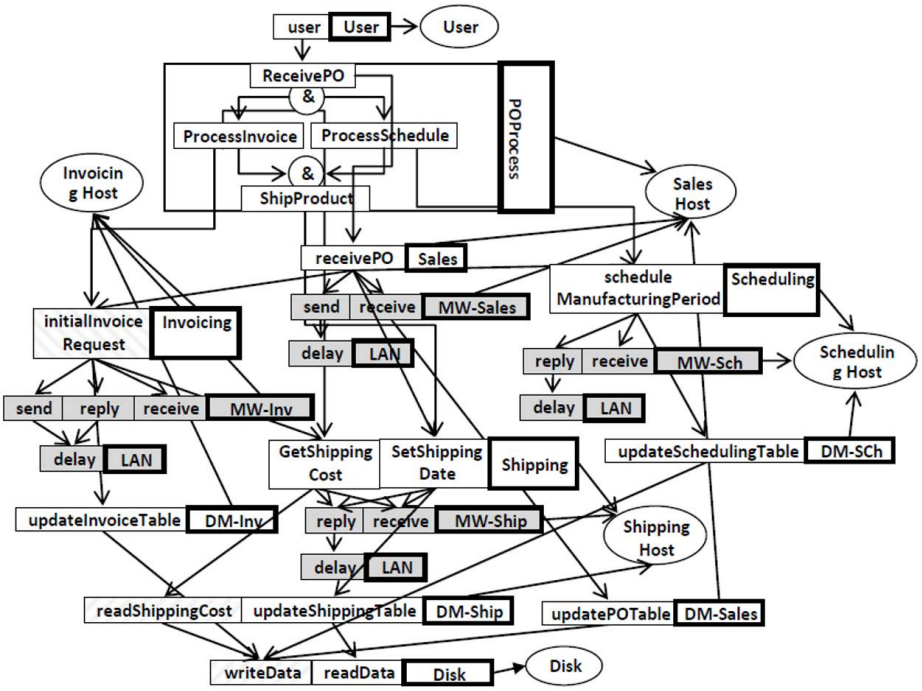

Fig. 14. LQN model of Purchase Order system (PSM)

in the LQN metamodel to create elements that store the results. To meet the performance requirement defined for the PO workflow (required mean response time <= 3s), three iterations of the model transformation chain have been executed as described in Table 1 for *$Nusers* = 100.

The first iteration represents the base case, where the multiplicity of all tasks and hosts equals one. The response time and throughput are calculated and passed to the *POSystem:Activity* through the trace-link *perfResultsTL: OutputResultTypeActivityTL*. Based on the LQN solver results, the *Sales:Task* is found to be the bottleneck server.

This is a case of software bottleneck, usually solved by increasing the concurrency level by having more threads in the pool. Thus we change the multiplicity of Sales to 15. The new value is propagated through the trace-links to their corresponding element in UML and CSM. The same procedures happen in the second iteration, except that the bottleneck is the processor *InvoicingHost:Node*. We increase its multiplicity value to 5 (which means using 5 cores instead of one). In the third iteration, the calculated response time is less than 3s, so it meets the performance requirement. More iterations could be done to find out if the requirement can be met with fewer resources.

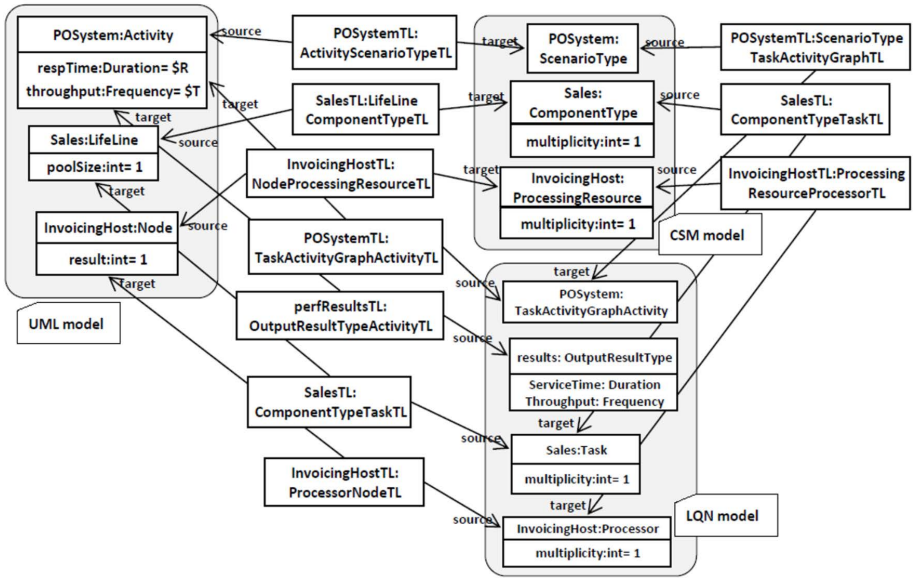

Fig. 15. Sample of traceability model of PO system

Table 1. Performance analysis of PUMA4SOA

$Nuser= 100, respTime = ((3,s, mean), req)			
	UML	CSM	LQN
1	POSystem:Activity {respTime = $R} *After feedback $R=7014ms*	POSystem: ScenarioType	POSystem: TaskActivityGraph results: OutputResultType {serviceTime = 7014ms}
	Sales: LifeLine {poolSize = 1}	Sales: ComponentType {multiplicity = 1}	Sales: Task {multiplicity = 1} **Solve software bottleneck -> multiplicity=15**
	InvoicingHost: Node { resMult = 1}	InvoicingHost: ProcessingResource {multiplicity= 1}	InvoicingHost: Processor {multiplicity = 1}
2	POSystem: Activity {respTime = 7014ms} *After feedback $R=3442ms*	POSystem: ScenarioType	POSystem: TaskActivityGraph results: OutputResultType {serviceTime = 3442ms}
	Sales:LifeLine { poolSize = 15}	Sales: ComponentType {multiplicity = 15}	Sales:Task { multiplicity = 15}
	InvoicingHost: Node { resMult = 1}	InvoicingHost: ProcessingResource {multiplicity = 1}	InvoicingHost: Processor { multiplicity = 1 } **Solve hardware bottleneck -> multiplicity=5**
3	POSystem: Activity {respTime=3442,ms} *After feedback $R=1607ms*	POSystem: ScenarioType	POSystem: TaskActivityGraph results: OutputResultType {serviceTime = 1607, ms}
	Sales: LifeLine {poolSize = 15}	Sales:ComponentType {multiplicity = 15}	Sales: Task {multiplicity = 15}
	InvoicingHost:Node { resMult = 5}	InvoicingHost: ProcessingResource {multiplicity = 5}	InvoicingHost: Processor { multiplicity = 5}

5 Related Work

Traceability information is used to manage the artifacts of a software system during its development life cycle. The survey in [1] has classified three traceability approaches: a) the *requirement-driven* approach which describe the traceability of requirement specification to its subsequent deployment and use; b) the *modeling* approach, which is mainly focused on defining tracing mechanisms for a modeling language at the metamodel level (such as special tracing relationships); and c) the *transformation* approach, where the tracing process is performed between a source and a target model during the model transformation, by creating trace-links between the source and the target model elements. PUMA4SOA uses the transformation approach to define trace-links between the elements of its three modeling languages, i.e. UML, CSM and LQN.

There are several papers in the literatures using the transformation approach; in [17] the author presented a method of attaching traceability generation code to pre-existing ATL programs [18]. The method produces a loosely coupled traceability, which can be used for any kind of traceability range and format. In [19] the authors presented a method of generating annotated models which contain traceability information, by merging the primary models with their defined trace models. The generated trace-links can be stored internally, where the trace-links are embedded as new elements inside the target models they refer to, or externally where the trace-links are stored separately in a new model. In [20] the authors proposed a traceability framework, implemented in the model-oriented language Kermeta, to facilitate modeling transformations. Using a trace metamodel, the framework allows for tracing the transformation chain within Kermeta. Model transformation trace-links are defined in the metamodel as a set of source nodes and target nodes.

Two ways to manage the complexity of traceability information in MDE where introduced in [16]. One is by identifying the trace-links through a process called Traceability Elicitation and Analysis Process (TEAP), which is mainly used to extract and analyze traceability relationships within an MDE process, to determine how these relationships would fit into a trace-link classification. The second way is by describing a strict metamodeling approach, which defines semantically rich trace-links between the elements of different models. Three characteristics are defined for the semantically rich trace-links: a) to be typed, b) to conform to a case-specific traceability metamodel, and c) to define a set of constrains within the case specific meatmodel to validate the requirements that cannot be captured by the metamodel itself. In this paper, we used a metamodeling approach similar to [16] to create a traceability metamodel for our PUMA4SOA, which defines trace-links between the elements of the UML, the CSM and the LQN elements. Also, in order to handle the problem of traceability loss when applying aspect-oriented techniques we extended the metamodel by defining new trace-links between the previously defined trace-links.

Similar to our work, in [21,22], the authors use trace-links between a software model (Smodel) and the corresponding Performance model (Pmodel) of a service-oriented system to study the change propagation when applying design patterns

to a Smodel. The purpose is to develop methods for incremental propagation of changes from Smodel to Pmodel, in order to study the performance effects of design patterns.

6 Conclusions

The paper focuses on defining a traceability metamodel for PUMA4SOA, which is used to define trace-links between different types of models, namely UML, CSM and LQN. We also addressed the problem of trace-links when applying aspect-oriented modeling techniques. Trace-links are used in PUMA4SOA to analyze the impact of changes at different model levels, i.e. UML, CSM and LQN, and to feed back the performance results from the LQN model to the UML model. We illustrate the proposed approach with a simple PO system.

In the future, we are planning to define formally the constraints within PUMA4SOA traceability meatmodel to express well-formedness rules that cannot be captured by the metamodel itself. We are also working on implementing the traceability metamodel proposed in this paper, integrating it with the existing PUMA4SOA model transformations.

Acknowledgements. This research was partially supported by the Natural Sciences and Engineering Research Council (NSERC) and industrial and government partners, through the hSITE Strategic Research Network.

References

1. Galvao, I., Goknil, A.: Survey of Traceability Approaches in Model-Driven Engineering. In: Proceedings of the 11th IEEE International Enterprise Distributed Object Computing Conference, pp. 313–326. IEEE Computer Society (2007)
2. Gotel, O.C.Z., Finkelstein, A.C.W.: An Analysis of the Requirements Traceability Problem. In: Proceedings of the International Conference on Requirements Engineering, pp. 94–101. IEEE Computer Science Press (1994)
3. Kolovos, D., Paige, R.F., Polack, F.A.C.: On-Demand Merging of Traceability Links with Models. In: From: 3rd ECMDA Traceability Workshop (2006)
4. Paech, B., von Knethen, A.: A Survey on Tracing Approaches in Practice and Research. Technical Report IESE Report Nr. 095.01/E, Fraunhofer - Institute of Experimental Software Engineering (2002)
5. Alhaj, M.: Automatic generation of performance models for SOA systems. In: Proceedings of the 16th International Workshop on Component-Oriented Programming (WCOP 2011), pp. 33–40. ACM (2011)
6. Alhaj, M., Petriu, D.C.: Approach for generating performance models from UML models of SOA systems. In: Proceedings of the 2010 Conference of the Center for Advanced Studies on Collaborative Research (CASCON 2010), pp. 268–282. IBM (2010)
7. Alhaj, M., Petriu, D.C.: Using Aspects for Platform-Independent to Platform-Dependent Model Transformations. International Journal of Electrical and Computer System (IJECS) 1(1), 35–48 (2012)

8. Woodside, C.M., Petriu, D.C., Petriu, D.B., Shen, H., Israr, T., Merseguer, J.: Performance by Unified Model Analysis (PUMA). In: Proceedings of the 5th International Workshop on Software and Performance (WOSP 2005), pp.1–12. ACM (2005)
9. Object Management Group: Unified Modeling Language Superstructure Version 2.2 formal/2009-02-02, http://www.omg.org/spec/UML/2.2/Superstructure/PDF
10. Object Management Group: UML Profile for Modeling and Analysis of Real-Time and Embedded Systems (MARTE) Version 1.1 formal/2011-06-02, http://www.omg.org/spec/MARTE/1.1/PDF
11. Object Management Group: Service oriented architecture Modeling Language (SoaML) formal/2012-03-01, http://www.omg.org/spec/SoaML/1.0/PDF
12. Petriu, D.B., Woodside, C.M.: An intermediate metamodel with scenarios and re-sources for generating performance models from UML designs. Software and Systems Modeling 6(2), 163–184 (2007)
13. Woodside, C.M., Neilson, J.E., Petriu, D.C., Majumdar, S.: The Stochastic Rendezvous Network Model for Performance of Synchronous Client-Server-like Distributed Software. IEEE Transactions on Computers 44(1), 20–34 (1995)
14. Woodside, C.M., Petriu, D.B., Siddiqui, K.H.: Performance-related Completions for Software Specifications. In: Proceedings of the 24th International Conference on Software Engineering (ICSE 2002), pp. 22–32. ACM (2002)
15. Tawhid, R., Petriu, D.C.: Automatic Derivation of a Product Performance Model from a Software Product Line Model. In: Proceedings of the 2011 15th International Software Product Line Conference (SPLC 2011). IEEE Computer Society (2011)
16. Paige, R.F., Drivalos, N., Kolovos, D.S., Fernandes, K.J., Power, C., Olsen, G.K., Zschaler, S.: Rigorous identification and encoding of trace-links in model-driven engineering. Software and Systems Modeling (SoSyM) 10(4), 469–487 (2011)
17. Jouault, F.: Loosely Coupled Traceability for ATL. In: Traceability Workshop at European Conference on Model Driven Architecture (ECMDA-TW), pp. 29–37 (2005)
18. Jouault, F., Kurtev, I.: Transforming Models with ATL. In: Bruel, J.-M. (ed.) MoDELS 2005. LNCS, vol. 3844, pp. 128–138. Springer, Heidelberg (2006)
19. Kolovos, D.S., Paige, R.F., Polack, F.A.C.: Merging Models with the Epsilon Merging Language (EML). In: Wang, J., Whittle, J., Harel, D., Reggio, G. (eds.) MoDELS 2006. LNCS, vol. 4199, pp. 215–229. Springer, Heidelberg (2006)
20. Falleri, J., Huchard, M., Nebut, C.: Towards a Traceability Framework for Model Transformations in Kermeta. In: Traceability Workshop at European Conference on Model Driven Architecture (ECMDA-TW), pp. 31–40 (2006)
21. Mani, N., Petriu, D.C., Woodside, C.M.: Propagation of Incremental Changes to Performance Models due to SOA Design Pattern Application. In: Proceedings of the International Conference on Software Engineering (ICPE 2013) (2013)
22. Mani, N., Petriu, D.C., Woodside, C.M.: Studying the Impact of Design Patterns on the Performance Analysis of Service Oriented Architecture. In: Proceedings of the 2011 37th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA 2011), pp. 12–19. IEEE Computer Society (2011)