From PIM and PSM with Performance Annotation to Palladio Models

Dariusz Gall[a]
Wrocław University of Science and Technology, Wybrzeże Wyspiańskiego 27, Wrocław, Poland

Keywords: SPE, Software Performance Engineering, Performance, PCM, Palladio, MDA, Model-driven Architecture, PIM, Platform-independent Model, PSM, Platform-specific Model, UML, Model Transformation.

Abstract: Essential step in Software Performance Enginnering (SPE) is finding software performance characteristics. We provide transformations of Platform Independent Model (PIM), and Platform Specific Model (PSM) models enriched by performance annotation in Modeling and Analysis of Real-Time Embedded Systems (MARTE) to the Palladio Component Model (PCM) performance model. The system's structural viewpoint, i.e., PSM's class and deployment diagrams, are mapped to the Component Repository. The system's behavior, i.e., sequence diagrams, are transformed into the PCM Service Effect Specifications. The PCM Resource Environment is generated from PSM's deployment diagram. Finally, the PCM Usage Model is created, combining these models and the PIM's use cases.

1 INTRODUCTION

Software systems development tends towards automating functional requirements implementation. It often refers to the model-driven development approaches, including Model-Driven Architecture (MDA) (Object Management Group, 2014). Software development, apart from functional requirements, should meet various quality requirements. One of the quality requirements is performance requirements, determining, for example, the expected system throughput, system response times, transaction processing times. According to Software Performance Engineering (SPE) (Smith and Williams, 2002), performance requirements, like other requirements, should be validated during the development process by constructing and evaluating performance models against the requirements. These models can be input into the Palladio (Reussner et al., 2016), the approach for resolving software system performance characteristics.

We include performance requirements in software development based on the MDA approach. In particular, we discuss how to construct the Palladio performance models of software systems. The presentation of the proposed approach is informal. An example illustrates the considerations.

Papers (Hahner et al., 2021) (Reiche et al., 2021) (Mazkatli et al., 2020) (Kroß and Krcmar, 2017) discuss the model-driven based construction of Palladio Component Model (PCM) models. However, they discuss generating PCM from existing implementations and only for selected software systems types. The idea of joining model-driven development and performance engineering is discussed in papers: (Ameller et al., 2021) (Cortellessa et al., 2007a) (Cortellessa et al., 2007b). The authors also discuss this topic in papers: (Walkowiak-Gall and Gall, 2015) (Gall and Huzar, 2010) (Chudzik et al., 2011). In particular, the papers propose an MDA-based approach incorporating performance requirements.

The paper structure is as follows. Section 2 presents the Platform Independent Model (PIM) and the Platform Specific Models (PSM), underlining performance aspects. In addition, we introduce the Palladio approach and the PCM models. In Section 3, the transformation between PIM, PSM and PCM models are discussed, together with some illustrative examples. The paper is concluded in Section 4 by debating the results and stating future work.

2 MDA AND PALLADIO

The MDA is a sketchily defined approach. We assume that functional and performance specifications are provided at the PIM level model and are transformed into PSM implementation level models, having made necessary architecture and design decisions,

[a] https://orcid.org/0000-0002-0685-1338

Gall, D.
From PIM and PSM with Performance Annotation to Palladio Models.
DOI: 10.5220/0011090900003176
In *Proceedings of the 17th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2022)*, pages 528-535
ISBN: 978-989-758-568-5; ISSN: 2184-4895

including execution environment selection and performance quality characteristics of the environment (Walkowiak-Gall and Gall, 2015) (Gall and Huzar, 2010). Without sacrificing the generality of the paper, we propose the following definitions of PIM models and PSM models. Additionally, we picked Java for the implementation platform.

2.1 Platform Independent Model

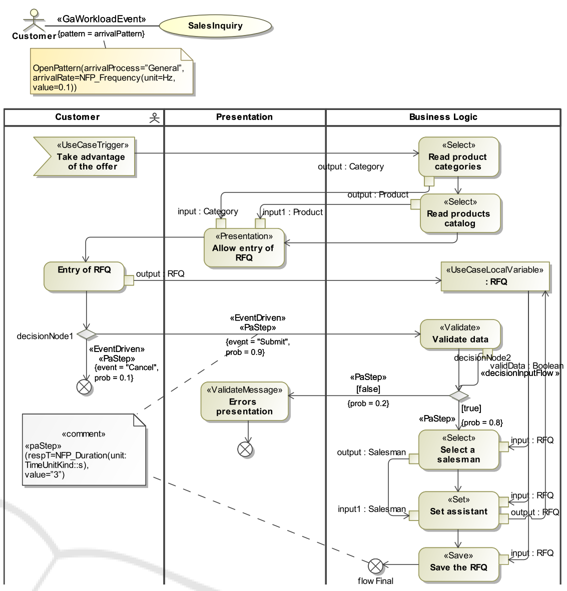
The PIM is a system specification that abstracts from implementation details. A proposal of the PIM is provided in the paper (Walkowiak-Gall and Gall, 2015), in the form of a UML Profile. The PIM model consists of following. The information model defines the structure of information processed within the system, which is expressed by elements of a class diagram (Object Management Group, 2017).

The use-case model defines use-cases and actors, i.e., functionalities of the system and users who use those functionalities, see Fig 1. A functionality is specified by use-case scenarios. An activity specifies use-case scenarios. Activity is divided into partitions, representing participants of the behavior and group actions for which they are responsible, Fig 1. Partitions used in the presented approach represent the actor's actions, the behavior of a system presentation layer, and the behavior of a system logic layer.

The above PIM proposal addresses only the functional aspect of PIM-level specification. Next, the PIM-level specification of performance characteristics is discussed. These characteristics are provided using the UML language's Modeling and Analysis of Real-time and Embedded profile (MARTE) (Object Management Group, 2019), supporting the modeling of real-time and embedded systems. Performance requirements are defined for a pair of an actor and a use case. For a given actor and a use case, an intensity of use case invocations is defined by the «gaWorkload-Event» stereotype. Additionally, it defines an arrival pattern of the use-case calls, either open workload - OpenPattern or closed workload - ClosedPattern.

In addition, an annotation with probabilities is set for each activity diagram representing a given use case. Probabilities determine the relative frequency of execution of a given group of scenarios. These annotations are attached to where the use-case scenarios separate, i.e., to outgoing control nodes' decision flows. Such annotation is marked with the «paStep» stereotype and a probability of transiting to annotated control flow.

Another aspect is data exchange between an actor and a system. If characteristics of data sent or received, e.g., data size, during actor-system interactions are essential for performance analysis, it is possible to annotate such exchange with these characteristics. Such information is stored in the message size parameter of «paCommStep» annotation.



«GaWorkloadEvent»
Customer
{pattern = arrivalPattern}
SalesInquiry
OpenPattern(arrivalProcess="General", arrivalRate=NFP_Frequency(unit=Hz, value=0.1))
Customer
Presentation
Business Logic
«UseCaseTrigger» Take advantage of the offer
«Select» Read product categories
output : Category
output : Product
«Select» Read products catalog
input : Category
input1 : Product
«Presentation» Allow entry of RFQ
Entry of RFQ
output : RFQ
«UseCaseLocalVariable» : RFQ
«EventDriven» «PaStep» {event = "Submit", prob = 0.9}
decisionNode1
«EventDriven» «PaStep» {event = "Cancel", prob = 0.1}
«Validate» Validate data
decisionNode2
validData : Boolean «decisionInputFlow»
«PaStep» [false] {prob = 0.2}
«ValidateMessage» Errors presentation
[true] «PaStep» {prob = 0.8}
«comment» «paStep» (respT=NFP_Duration(unit: TimeUnitKind::s), value="3")
output : Salesman
«Select» Select a salesman
input : RFQ
input1 : Salesman
«Set» Set assistant
output : RFQ
«Save» Save the RFQ
flow Final

Figure 1: PIM example.

Last but not least is to provide expectations in time duration execution for actor-system interactions. They are defined by time duration constraints on these interactions in annotations with a time expression, expressed by «paStep». Moreover, those interactions become points of measure to verify performance characteristics computed during the performance analysis against the provided expectations.

2.1.1 Example

The Sales system is an illustrative example. Due to space limitations, it is presented fragmentarily, one actor and one use case representing the sales inquiry functionality. The Fig 1 shows use-cases and associated actors. The actor Customer is associated with the use-case SalesInquiry, i.e., it initiates this use case. The association of actor and use-case is annotated by the «gaWorkloadEvent» stereotype OpenPattern, which indicates open workload intensity with general distribution at an arrival rate of 0.1 per second.

The activity in Fig 1 is the use-case specification. Most actions are related to system behavior, like data fetching, data processing. These actions as a result of transformation will turn into implementation level calls, including implementation platform calls. These calls will induce demands on a CPU, disks, and other resources. The activity has performance characteristics. For example, «paStep» applied to outgoing flows of the decision node "decisionNode1" in an actor partition states the probabilities that the actor will send (probability=0.9) or cancel (probability=0.1) a sell query. Similarly, «paStep» is applied to "decisionNode2"'s outgoing flows in the business logic partition. Worth mentioning is that activity within this partition is transformed into PSM-level. Thereby, to do performance analysis, this characteristic has to be mapped into PSM-level as well. Finally, the model Fig 1 contains the «paStep» model element spanned between the "decisionNode1"'s output control flow "Send" and "flowFinal." This «paStep» is an example of asserting the expected time duration between two moments: a Seller actor presses "Send," and the activity is finished. It states that the meantime of interaction durations has to be 3 seconds.

2.2 Platform Specific Models

Two models are considered at the PSM level: PSM Specification (PSM-Spec) and PSM Instance (PSM-Inst) (Walkowiak-Gall and Gall, 2015). The PSM-Spec model represents the PSM-Spec architecture of the system - it defines the types of system nodes together with the types of components arranged in them, while the PSM-Inst model is the PSM-Inst architecture, i.e., an instance of the PSM-Spec architecture.

The PSM-Spec model in UML is expressed in a class diagram (design classes), type-level deployment diagrams, and sequence diagrams. Design classes result from transforming an entire PIM model. The system designer elaborates the transformation, usually based on architectural patterns. Design classes are arranged in nodes as reflected in the deployment diagram. Sequence diagrams in the PSM-Spec model result from the transformation of activity diagrams from the PIM model; they follow the architectural design pattern used.

As stated before, the PSM-Spec has structural elements, which are elements present on a class diagram: packages, classes, interfaces, see Fig. 2. They correspond to Java packages, classes, interfaces. Each operation always has one return parameter and can have zero or more input parameters. There are more constraints, but they are omitted for the sake of brevity. Since the PSM-Spec is at the implementation level, it refers to elements of the implementation platform, i.e., the Java platform. Thereby, it is necessary to include a Platform Model (PM), i.e., a model containing packages, classes, and interfaces representing Java APIs, libraries, frameworks.

Behaviors of the PSM-Spec are defined by interactions (sequence diagrams) (Object Management Group, 2017). Each non-abstract operation has to point to a behavior, which corresponds to the method definition in Java. A single sequence diagram defines this behavior according to the below constraints. The interaction consists of a "self" lifeline representing an object, or a class in case of a static operation, being callee of an operation. The self lifeline defines a sequence of messages corresponding to a sequence of statements in the Java method. Only interactions between the self and other lifelines are shown. See Fig 3.

There are three sets of performance characteristics in the PSM-Spec. First is created by mapping all PIM elements with performance annotation into PSM-Spec elements, tracing a design and implementation (transformation) process.

The second is a collection of the characteristics included in the PM model. These are stated for platform APIs' operations, frameworks' behaviors and are provided by the notion of «paStep» from the MARTE profile. The PM model's performance characteristics are specific to an implementation platform and parametrically dependent on the execution environment. They are obtained in the process of platform performance analysis, performance experiments (Gall and Huzar, 2010) (Chudzik et al., 2011).

Finally, the PSM-Inst is complemented by the performance characteristics of nodes and connections between nodes. A node instance's computational power is defined by the stereotype «gaExecHost» and throughput parameter. The notion of «gaCommHost» defines a connection's instance bandwidth, capacity, and network latency (blocking time) parameters.

2.2.1 Example

The interactions on Fig 3 and deployment diagram, see Fig 2, are part of a PSM-Spec model, resulting from the transformation of the example PIM, discussed in Section 2.1.

The interactions presents a portion of the SalesInquiry use-case implementation, including invocations of platform libraries' operations, assigning variable value. Both interactions are related to operations, which are placed in design classes Fig 3. It is essential to notice that the interaction has decision node "decisionNode2", which is mapped from "decisionNode2" of the PIM's SalesInquiry activity in Fig 1.

Diagram in Fig 2a presents two connected nodes with deployed artifacts. Those artifacts represent the design classes of the PSM-Spec, see Fig 2c, in which binary versions are situated within a given node.

The example PSM-Inst is shown in Fig 2b. It contains a deployment diagram with nodes' instances and a connection instance. The nodes' instances are annotated by «gaExecHost» elements stating their

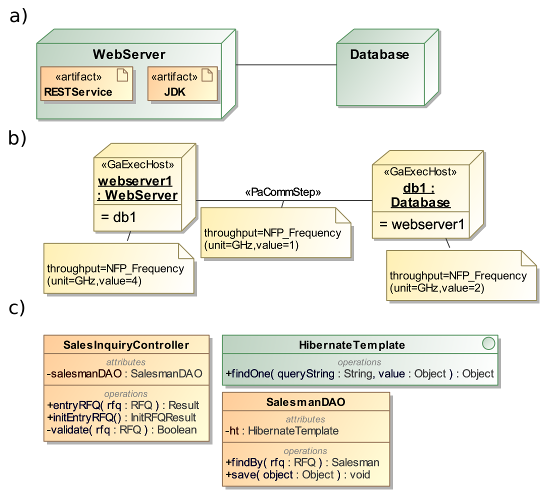

a)
WebServer
«artifact» RESTService
«artifact» JDK
Database
b)
«GaExecHost» webserver1 : WebServer
= db1
«PaCommStep»
«GaExecHost» db1 : Database
= webserver1
throughput=NFP_Frequency (unit=GHz,value=4)
throughput=NFP_Frequency (unit=GHz,value=1)
throughput=NFP_Frequency (unit=GHz,value=2)
c)
SalesInquiryController
attributes
-salesmanDAO : SalesmanDAO
operations
+entryRFQ(rfq : RFQ) : Result
+initEntryRFQ() : InitRFQResult
-validate(rfq : RFQ) : Boolean
HibernateTemplate
operations
+findOne(queryString : String, value : Object) : Object
SalesmanDAO
attributes
-ht : HibernateTemplate
operations
+findBy(rfq : RFQ) : Salesman
+save(object : Object) : void

Figure 2: Example of: a) Deployment specification - PSM-Spec b) Deployment instance - PSM-Inst c) Design classes - PSM-Spec.

performance characteristics. Similarly, the connection instance is also annotated by the notion «paCommHost», providing network connection speed characteristics. Important to mention that the example PSM-Inst and PSM-Spec represent the system's instance, which can be tested for performance.

2.3 Palladio Models

The Palladio approach provides PCM, a modeling language for documenting software architecture (Reussner et al., 2016) (Reussner et al., 2011). The PCM model is an input for various tools predicting the provided system's quality characteristics.

The PCM describes an architecture from viewpoints essential for making quality properties analysis possible. A structural viewpoint represents the static properties and consists of the repository and assembly view types. A behavioral viewpoint specifies the dynamics of the system. In addition, it contains quality characteristics such as information on workload posed on the software system by actors, the specification of data provided by actors, like size, structure, and probabilities of various actors' behaviors. The usage model view type is a set of usage scenarios, defining scenario behavior and workload. Finally, resource environment view type and allocation view type constitute the deployment viewpoint of the PCM. The resource environment view type contains resource containers and linking resources. The allocation view type provides information on how components instances, i.e., assemblies contexts, are assigned to the resource containers.

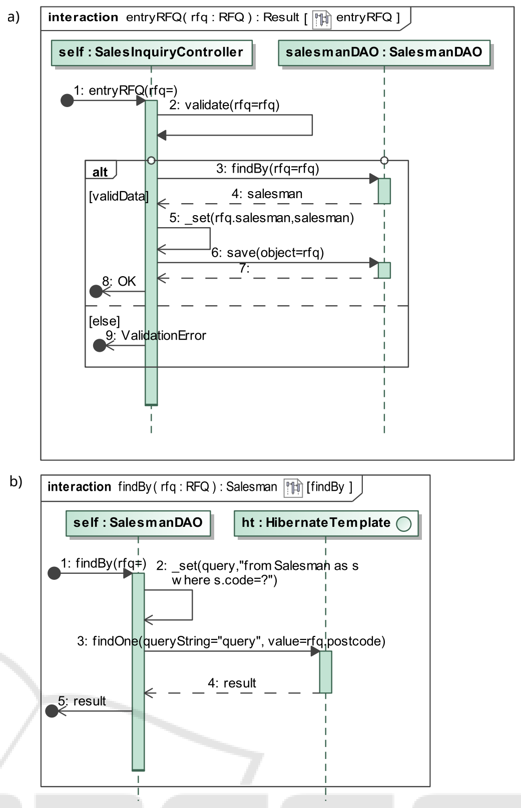

a)
interaction entryRFQ(rfq : RFQ) : Result [entryRFQ]
self : SalesInquiryController
salesmanDAO : SalesmanDAO
1: entryRFQ(rfq=)
2: validate(rfq=rfq)
alt
[validData]
3: findBy(rfq=rfq)
4: salesman
5: _set(rfq.salesman,salesman)
6: save(object=rfq)
7:
8: OK
[else]
9: ValidationError
b)
interaction findBy(rfq : RFQ) : Salesman [findBy]
self : SalesmanDAO
ht : HibernateTemplate
1: findBy(rfq=)
2: _set(query,"from Salesman as s where s.code=?")
3: findOne(queryString="query", value=rfq.postcode)
4: result
5: result

Figure 3: Examples of implementation methods - PSM-Spec: a) entryRFQ() from SalesmanController class, b) findBy() from SalesmanDAO class.

3 GENERATING PALLADIO MODELS FROM PIM AND PSM

A sequence of transformations generates the PCM models. Each transformation is responsible for generating a model of a single view type. The first transformation maps the deployment diagram of PSM models to the PCM Resource Environment. The next step is to generate PCM components from PIM and PSM models with behaviors and store them in Repository. Subsequently, an assembly of components is turned into a system assembly resulting from the transformation of PIM, PSM, and PCM Repository models into a PCM System. The system assembly is finished by transforming the PSM models and the PCM System into the PCM Allocation, which allots PCM assemblies to resource environments. Finally, PIM, PSM, and PCM System models are used to generate PCM Usage, i.e., provide users, workloads, and scenarios to complete the system documentation.

Below, these transformations are defined and discussed. These definitions are provided using structured programming pseudocode combined with mathematical notation. For the sake of readability, some of these transformations are expressed using auxiliary functions and helper transformations. It is assumed after that that the following functions are defined.

Function $type(e)$ returns a meta-class for model element e. Function $tag(e,s,t)$ returns a tag's value t of a stereotype s applied to model element e. There are also functions returning all model elements of a given meta-class in a given model. For example, function $Classes(M)$ returns a set of all UML classes in a model M.

The following function introduces transformation tracing, i.e., information about which target elements were created from which source elements. It is used to avoid tight coupling among transformations and to make them easier to understand and maintain (Falleri et al., 2006) (Hassane et al., 2020). The function $from(e)$ traces a source element from which the element e was transformed.

The first transformation generates PCM Resource Environment. The transformation maps PSM-Inst UML deployment elements into the PCM Resource Environment model. It also creates communication links between these resources. For each instance node in PSM-Inst, a resource container is created and added to PCM Resource Environment. The transformation is straightforward, and for brevity, the transformation pseudo-code is not presented in the paper.

The next transformation creates and adds components to the PCM Repository model. Each UML artifact deployed in PSM-Spec nodes is mapped to a PCM basic component and provided interfaces. PCM operations of the provided interfaces result from UML operations of classes represented by an artifact. However, only UML operations, which are called from operations of classes represented by other artifacts, are mapped to PCM operations since the other artifacts are also mapped to basic components.

The helper function $isInvoked(op,art)$ verifies whether a UML operation is called within a given artifact. It checks if an operation op invocation exists within UML interactions of classes' operations manifested in the artifact art. Complementary to these mappings, the transformation sets all interfaces required by the basic components.

This transformation creates also interfaces with operations provided by the PCM system. These PCM operations are called from usage scenarios of the PCM usage model during the system performance examinations. Again, PCM operations transform from operations in the PSM-Spec. A UML operation is mapped to PCM when it is the very first call when the system receives a request (on the system's boundary). Such UML operation is found by tracing a UML activity mapping, defining a use-case at the PIM level, into the PSM implementation level, particularly by tracing a control flow transition between an actor and a presentation or a business logic partition.

The helper function $cfActorSystem(act)$, for a given UML activity act, provides control flows, in which the source node is in an actor partition, and the target node is in a presentation or a business logic partition. In other words, it provides control flows which cross the system boundary. An assumption is made that it is possible to trace a mapping from the control flow to an entry point message (UML found message) in the UML operation's behavior (interaction) for each such control flow. Thereby UML operations, which are transformed to PCM system's operations, are selected. Please refer to the transformation *PSM_Deployment_Artifacts_to_PCM_Repository* below.

```
PSM_Deployment_Artifacts_to_PCM_Repository
input: PIM, PSMSpec: UML
output: Rep: Repository

for each art in Artifacts(PSMSpec) do
 - Create a basic component and provided
 interface
 - Get all operations found in artifact's
 classes and check whether they have to be in
 the provided interface
  for each op in $\{o \in \bigcup_{cls \in art.manifestation} cls.operation \mid \exists a \in (artifacts - \{art\}) (isInvoked(op,a))\}$ do
    - Add op to provided interface
    - Get all interfaces called by the op
    for each art' in $\{a \in (artifacts - \{art\}) \mid \exists o \in \bigcup_{cls \in art.manifestation} cls.operation( isInvoked(op,a))\}$ do
      - Set an interface of art's as provided
      interface of the basic component.
    - Create a SEFF behavior of the op
    PCM_Signature_to_PCM_SEFF(op, provInter)

 - Check if a system actor calls any operations
 usersOpers:=$\left\{o \in \bigcup_{cls \in art.manifestation} cls.operation \mid o.method \in \bigcup_{\substack{uc \in UseCases(PIM) \\ cf \in cfActorSystem(uc.classifierBehavior)}} \{from(cf).interaction.specification\}\right\}$
 - If yes, then
 if |usersOperation| > 0 then
  - Create provided interfaces with operations
  for users.
  for each op in usersOperations do
   - Add op to provided interface
   - Create a SEFF behavior of the op
   PCM_Signature_to_PCM_SEFF(op, provInter)
```

For each PCM operation realized by a basic component, a service effect specification (SEFF) behavior is generated. The auxiliary transformation creates a PCM SEFF behavior for a given PCM operation from UML interactions. It starts by tracing the UML operation from which the PCM operation was generated. Next, by calling another helper transformation, a UML interaction of the operation is mapped fragment by fragment. Please refer to the auxiliary transformation *PCM_Signature_to_PCM_SEFF* below.

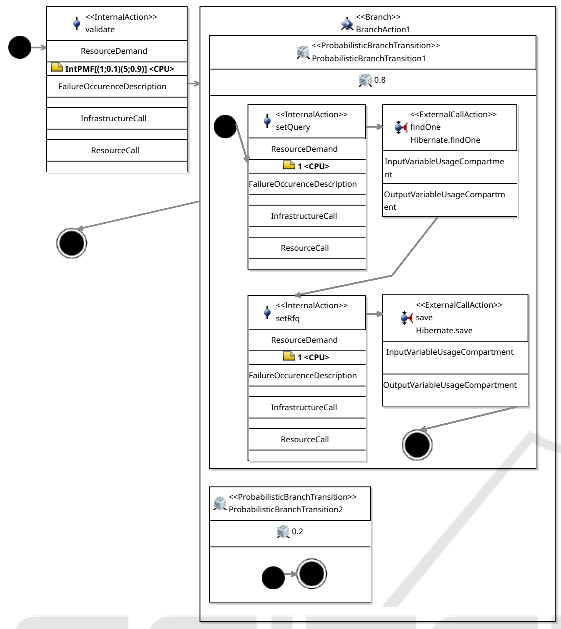
```
PCM_Signature_to_PCM_SEFF
input: opSig: PCM::OperationSignature
 provInter: PCM::ProvidedInterface
output: seff: ResourceDemandingSEFF

- Get original UML operation and method
meth:=from(opSig).method
- Create an SEFF, StartAction, StopAction.
- Generate a SEFF for the UML interaction
PSM_Interaction_to_PCM_SEFF(
meth.fragment,startAction,seff,provInter.comp)
```

This transformation traverses a sequence of UML Interaction's fragments, and depending on the type of interaction, provides proper mapping. A message occurrence corresponding to setting the value of a variable is mapped to PCM internal action. A UML call operation occurrence specification is transformed into PCM external call action if the called operation corresponds to any operation in the provided interfaces. The call action is set with references to the required interface and the operation and actual parameters. However, when the called operation is not found in any provided interfaces, a sequence diagram of the operation has to be recursively transformed and merged into the parent SEFF.

For alternative UML combined fragments, the transformation creates PCM branch action, and for each UML operand a branch transition, either probabilistic or guarded. Next, for each operand, the transformation is called recursively, to transform interaction fragments embedded in the operands. Results are merged with a parent SEFF. Similarly, for loop combined fragment, a PCM LoopAction is created. Next, for the loop's operand, the transformation is called recursively and results are merged. Please refer to the auxiliary transformation *PSM_Interaction_to_PCM_SEFF* below.

```
PSM_Interaction_to_PCM_SEFF
input:
 fragments: sequence of
 UML::InteractionFragment
 curNode: PCM::AbstractAction
 behavior: PCM::ResourceDemandingBehaviour
 owner: PCM::BasicComponent
output: lastNode: PCM::AbstractAction

for each frag in fragments do
 - When type(frag) is an alternative,
 create the PCM branch action.
  - For each operand io, create the PCM branch
  and set either guard or probability.
  - Generate a SEFF for the fragment
  PSM_Interaction_to_PCM_SEFF
  (frag.io.fragment,startAction,branch,owner)

 - When type(frag) is a loop, create
  the PCM loop action
  - Retrieve tag(frag.operand,«paStep»,
  repetitions) and set the repetitions' number
  - Generate a SEFF for the fragment
  PSM_Interaction_to_PCM_SEFF(
  frag.operand.fragment,startAction,loop,owner)

 - When type(frag) is a UML operation call,
 and the UML operation is not mapped to
 PCM operation.
  - Generate a SEFF for the fragment and
  merge it with the owner SEFF.
  node:=PSM_Interaction_to_PCM_SEFF(frag.
  message.signature.method.fragment,curNode,
  behavior ,owner)

 - When type(frag) is a UML operation call,
 and the UML operation is mapped
 to PCM operation.
  - Do call to the mapped PCM operation:
  from(frag.message.signature).

 - When a value is assigned to a variable.
  - Add PCM internal action.

 curNode.successor := node
 curNode := node
 behavior.steps += node

lastNode := curNode
```

An example of applying this transformation in Fig 3 is shown in Fig 4. The UML operation in Fig 3a has its counterpart in a provided interface of a PCM basic component stored in the PCM Repository model (not shown). Since it is provided operation, a SEFF is generated. For example, an alternative combined fragment from Fig 3a is mapped to PCM branch action in Fig 4. Another example is the UML call operation, i.e., call of behavior, transformed into sequence actions in Fig 4. It is worth noticing that since the called operation, Fig 3b is not mapped into the PCM operation, its transformation result is merged in the SEFF. The last example is the mapping of UML call operation to API operation. Since it is a call to an external component representing API, it is mapped to the PCM external call action.



<<InternalAction>> validate
ResourceDemand
IntPMF[(1;0.1)(5;0.9)] <CPU>
FailureOccurenceDescription
InfrastructureCall
ResourceCall
<<Branch>> BranchAction1
<<ProbabilisticBranchTransition>> ProbabilisticBranchTransition1
0.8
<<InternalAction>> setQuery
<<ExternalCallAction>> findOne Hibernate.findOne
InputVariableUsageCompartment
OutputVariableUsageCompartment
1 <CPU>
<<InternalAction>> setRfq
<<ExternalCallAction>> save Hibernate.save
<<ProbabilisticBranchTransition>> ProbabilisticBranchTransition2
0.2

Figure 4: PCM SEFF example.

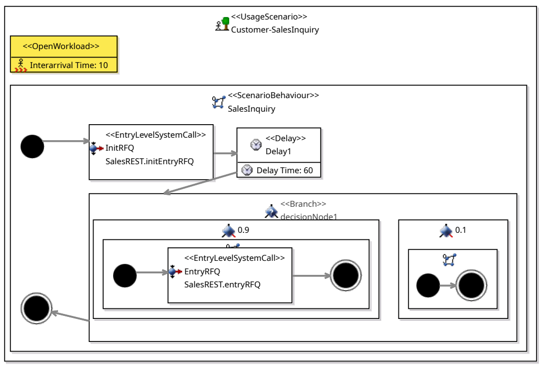
Subsequent transformation creates a PCM system model. It adds a PCM system assembly entity and system-provided interfaces, which publish operations invoked by a scenario behavior in a usage model. Next, the system assembly is filled with components' assembly contexts connected via components interfaces. An assembly context of a component requiring an interface is connected with an assembly context of a component providing the interface. There is always only one possible connection since only one component provides a given interface. The transformation is straightforward, and for brevity, the transformation pseudo-code is not presented in the paper.

Succeeding transformation creates a PCM allocation model, which has information on the assemblies' allocation to resource containers. The allocation is made concerning the deployment specification provided in the PSM-Inst model. The transformation is straightforward, and for brevity, the transformation pseudo-code is not presented in the paper.

Finally, UML PIM's use-cases are transformed into a PCM usage model. A new PCM Usage scenario is added for each pair of an actor and a use-case, it contains a workload specification and a scenario behavior. Because of the space limits, the transformation pseudo-code is not presented in the paper.

The scenario behavior is generated by moving through a use-case's activity, from node to node (action nodes, decision nodes) via edges within an actor's partition. An action node within the actor's partition is mapped to PCM delay (thinking) action in the PCM scenario. For a decision node, a corresponding PCM Branch node is added, and its alternative sub-scenarios are filed by mapping for each outgoing flows of the decision node. A PCM entry-level system call of corresponding (through tracing) operation is made when there is a transition outside the actor partition. Next, the graph continued within system partitions is explored to check if and in which places it transits back to the actor partition. If there is no return, then it is assumed that the scenario is finished. When there is a single return to a node within the actor's partition, the movement through nodes and edges within the partition is continued, starting from the return node. When it returns to multiple nodes, a PCM branch node is added, and for each return, an alternative scenario is created. For the sake of simplicity, we are considering only acyclic graphs. Because of the space limits, the transformation pseudo-code is not presented in the paper.

An example of applying this transformation on Fig 1 is shown in Fig 5. Without losing generality, only PCM scenario behavior is discussed. The first node of the UML activity in Fig 1 is connected with a node outside the actor's partition, which is mapped to a PCM entry-level system call. There is only one return to the actor's partition; thereby, the next node in the partition is mapped. It is a UML action; hence it is transformed to PCM delay action. The following is a decision node; consequently, it is mapped to PCM branch action. Two PCM branch transitions are added. One includes a PCM entry-level system call, and the other does nothing, i.e., end scenarios.



<<UsageScenario>> Customer-SalesInquiry
<<OpenWorkload>>
Interarrival Time: 10
<<ScenarioBehaviour>> SalesInquiry
<<EntryLevelSystemCall>> InitRFQ SalesREST.initEntryRFQ
<<Delay>> Delay1
Delay Time: 60
<<Branch>> decisionNode1
0.9
0.1
<<EntryLevelSystemCall>> EntryRFQ SalesREST.entryRFQ

Figure 5: PCM Usage Scenario Behavior - an example.

4 CONCLUSIONS

We discuss Palladio's performance models generation from MDA models, i.e., PSM and PIM. The PIM is a platform-independent functional specification in UML and is extended by performance characteristics in the UML MARTE Profile. The PSM models, given in the UML, provide implementation details and have performance characteristics of deployment nodes in MARTE.

We introduce the transformations of such PIM and PSM models into PCM models, i.e., PCM Resource Environment, Repository, System, Allocation, and Usage models. However, the transformations do not fully transform performance characteristics, e.g., PCM parameters and data size are not considered, neither mapping MARTE performance statements into PCM StoEx is. The transformation provides structure and behavior documentation of a system in the PCM, which has default performance characteristics.

Future work is to enhance the transformation by mapping PIM and PSM performance characteristics into PCM. In particular, MARTE statements have to be mapped to PCM StoEx statements. Moreover, the transformation should derive some performance characteristics, e.g., generate a loop repetition number from data characteristics. Another task is to provide transformation support for concurrency programming. Finally, the transformation should be implemented.

REFERENCES

Ameller, D., Franch, X., and Gómez (2021). Dealing with non-functional requirements in model-driven development: A survey. *IEEE Transactions on Software Engineering*, 47(4):818–835.

Chudzik, K., Gall, D., and Huzar, Z. (2011). Szeregowanie zadań w oszacowaniu wymagań wydajnościowych modelu psm. In *Projektowanie, analiza i implementacja systemów czasu rzeczywistego : praca zbiorowa*, pages 17–28. Wydawnictwa Komunikacji i Łączności, Warszawa.

Cortellessa, V., Marco, A. D., and Inverardi, P. (2007a). Integrating performance and reliability analysis in a non-functional MDA framework. In *Fundamental Approaches to Software Engineering, 10th International Conference, FASE 2007*, volume 4422 of *Lecture Notes in Computer Science*, pages 57–71. Springer.

Cortellessa, V., Marco, A. D., and Inverardi, P. (2007b). Non-functional modeling and validation in model-driven architecture. In *2007 Working IEEE/IFIP Conference on Software Architecture (WICSA'07)*, pages 25–25.

Falleri, J.-R., Huchard, M., and Nebut, C. (2006). Towards a traceability framework for model transformations in kermeta.

Gall, D. and Huzar, Z. (2010). Wymagania przepustowości w transformacji pim do psm. In *Metody wytwarzania i zastosowania systemów czasu rzeczywistego : praca zbiorowa*, pages 105–116. Wydawnictwa Komunikacji i Łączności, Warszawa.

Hahner, S., Seifermann, S., Heinrich, R., Walter, M., Bureš, T., and Hnětynka, P. (2021). Modeling data flow constraints for design-time confidentiality analyses. In *2021 IEEE 18th International Conference on Software Architecture Companion (ICSA-C)*, pages 15–21.

Hassane, O., Mustafiz, S., Khendek, F., and Toeroe, M. (2020). A model traceability framework for network service management. New York, NY, USA. Association for Computing Machinery.

Kroß, J. and Krcmar, H. (2017). Model-based performance evaluation of batch and stream applications for big data. In *2017 IEEE 25th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 80–86.

Mazkatli, M., Monschein, D., Grohmann, J., and Koziolek, A. (2020). Incremental calibration of architectural performance models with parametric dependencies. *2020 IEEE International Conference on Software Architecture (ICSA)*.

Object Management Group (2014). Model Driven Architecture (MDA) Guide rev. 2.0. OMG Document. https://www.omg.org/cgi-bin/doc?ormsc/14-06-01.

Object Management Group (2017). The Unified Modeling Language. OMG Document. https://www.omg.org/spec/UML/.

Object Management Group (2019). UML Profile for MARTE. OMG Document. https://www.omg.org/spec/MARTE.

Reiche, F., Schiffl, J., and Weigl, A. (2021). Quantification of correctness with palladio and key: Case study data.

Reussner, R., Becker, S., Burger, E., Happe, J., Hauck, M., Koziolek, A., Koziolek, H., Krogmann, K., and Kuperberg, M. (2011). The palladio component model. Technical Report 14.

Reussner, R. H., Becker, S., Happe, J., Heinrich, R., Koziolek, A., Koziolek, H., Kramer, M., and Krogmann, K. (2016). *Modeling and Simulating Software Architectures: The Palladio Approach*.

Smith, C. U. and Williams, L. G. (2002). Performance solutions: A practical guide to creating responsive, scalable software. USA. Addison Wesley Longman Publishing Co., Inc.

Walkowiak-Gall, A. and Gall, D. (2015). Pim-psm pattern-aware transformations. In *From requirements to software : research and practice*, pages 101–117. Polish Information Processing Society, Warszawa.