# Message-Oriented Middleware for System Communication: A Model-Based Approach

Roland Petrasch(✉)

Department of Computer Science, Thammasat University, Bangkok, Thailand
roland.petrasch@cs.tu.ac.th

**Abstract.** Distributed systems with heterogenous platforms and communication components like IoT devices require message-oriented middleware (MOM). Protocol translation, message model handling, message queueing and conversion, security, transactional consistency, monitoring are examples for the features and aspects of MOM. This paper presents a model-based approach for the development of MOM components using the UML and UML Profiles for MOM and enterprise integration patterns. A model-to-model transformation is used for the preparation of the design model for code generation.

**Keywords:** Internet-of-Things · Process modeling · UML · Activity diagram · Model-driven-development · Model-driven architecture · Enterprise Integration Patterns · EIP · Message-oriented-Middleware · MoM

## 1 Introduction and Related Work

In the IoT era, message-oriented middleware (MOM) and machine to machine (M2M) communication play a crucial role: According to Gartner, in 2016, 6.4 billion of connected "things", i.e. IoT devices, worldwide will be in use – a rise of 30% compared to 2015 [1]. To meet the increasing demand for high-quality communication products, substantial research and development efforts have been made leading to significant advances in the discipline of communication technologies during the last decade [2], e.g. in the area of IT-security [3], or practical methodologies [4].

A plurality of IoT relevant norms, industrial (de facto) standards, and products exist, e.g. Advanced Message Queuing Protocol (AMQP) by OASIS [5] and ISO [6], Java Messaging Service by JCP [7] (JMS is part of the Java Enterprise API), and MQ Telemetry Transport (MQTT) by OASIS [8]. The integration of heterogeneous system components using these standards and communication products is possible via a middleware layer. Nowadays, concepts and implementations of MOM are well-accepted and well-established in practice. From a technological standpoint, MOM often comprise components like message and resource broker, transaction manager, persistence service/DBMS, request scheduler etc.

But the IoT paradigm also "raises a number of new challenges in the software engineering domain" [9]. Requirements, e.g. in the form of business or domain processes, need to transformed to architectural concepts and software design specification that take IoT aspects, e.g. mobility of IoT devices, security of IoT data, or performance, into account [10, 11]. New or advanced software architectures, modeling languages, and modeling methods are helpful to take these new IoT aspects into account and reduce the complexity of the different components.

This paper addresses some of the challenges: It presents a model-based approach for developing middleware for IoT and Enterprise applications with a focus on enterprise integration patterns (EIP) [12]. Existing approaches focus on certain aspects of middleware, e.g. [13, 14], but they do not take advantage of UML, UML Profile, EIP patterns, and/or M2M transformations.

## 2 Development of MOM: A Model-Based Approach

### 2.1 Introduction to Model-Based Software Development

The general pattern for model-based development uses a PIM (platform independent model, e.g. a domain model, that is transformed into a PSM (platform specific model) if the meta-models of the PIM and the PSM are not identical (otherwise it is called a PIM-PIM transformation). The PSM serves as an input (PIM) for the next transformation(s). Transformation types are model-to-model (M2M), model-to-text (M2T), or text-to-model (T2M). Typically, the last step of a forward-engineering approach is the code-generation (model-to-text transformation), so that a PSI (platform specific implementation, i.e. code) is created (s. OMG's Model Driven Architecture [15]). A PIM as an input or source artefact for a transformation is considered platform independent, because it conforms only to its own meta-model (language specification) and is independent from the meta-model of the output or target model (PSM). Therefore, a PIM-PSM transformation involves two meta-models (source and target MM).

As the modeling language, the GPML (general purpose modeling language) UML was chosen [16]. The UML meta-model (language specification) is MOF-conform [17] and provides a lightweight extension mechanism: A UML Profile extends UML meta-model elements by defining a set of Stereotypes so that new domain ortechnological aspects can be included in the modeling. The possibility to create a DSML (domain specific language) for MOM/IOT (heavy-weight approach) was evaluated, but at the end, the existing UML diagrams were considered sufficient.

Figure 1 gives an overview of the approach used in this paper: Domain models, like business processes (UML activity diagrams or BPMN models) and domain class diagrams are used as a starting point (PIM). A UML Profile for MOM/IoT is developed and applied. The system or software architect then prepares the models for the model-to-model transformation: Certain model elements can be marked with Stereotypes. The transformation result is a first software design model (PSM), e.g. class or component diagrams. With the EIP Profile (application of enterprise integration patterns), design models are prepared for code generation (model-to-text transformation).

Different code generators for each target platform exist, e.g. Java Enterprise application, Apache Camel routes for middleware services (MOM). Generated code artefacts (PSI) are used for further manual programming.

The tool chain is based on the Eclipse modeling project [18]: Papyrus (UML Profiling and Modeling), QVTo as the Operational QVT implementation (model-to-model transformations), and Acceleo as the MOFM2T implementation (code generation).

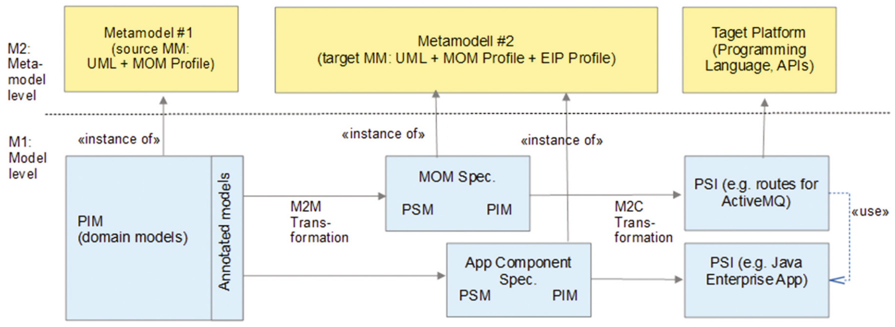

M2: Meta-model level
M1: Model level
Metamodel #1 (source MM: UML + MOM Profile)
Metamodell #2 (target MM: UML + MOM Profile + EIP Profile)
Taget Platform (Programming Language, APIs)
«instance of»
«instance of»
«instance of»
PIM (domain models)
Annotated models
M2M Trans-formation
MOM Spec.
PSM
PIM
App Component Spec.
PSM
PIM
M2C Trans-formation
PSI (e.g. routes for ActiveMQ)
PSI (e.g. Java Enterprise App)
«use»

**Fig. 1.** Model-based software development approach

## 2.2 Example PIM (Domain Model)

To exemplify the approach, an example scenario from the manufacturing domain is used: the quality control (QC) of a production material. Figure 2 shows a detail of the domain class model for the quality control (test) for a production material that usually produces a QC result dataset. A material can be tested several times.

The process is modeled as a UML activity diagram (Fig. 3): The QC procedure is initiated by a control app that activates a transport robot. The material is delivered to the QC unit where the material quality check takes place. The result of the QC procedure is sent to the transport robot and the control app.

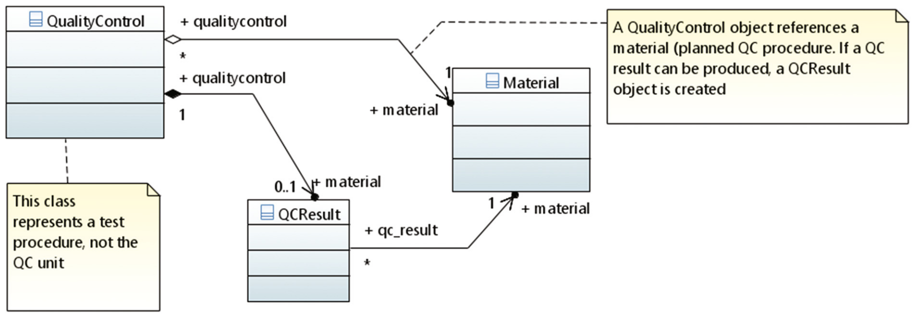

QualityControl
+ qualitycontrol
*
+ qualitycontrol
1
1
+ material
Material
A QualityControl object references a material (planned QC procedure. If a QC result can be produced, a QCResult object is created
0..1
+ material
QCResult
+ qc_result
*
1
+ material
This class represents a test procedure, not the QC unit

**Fig. 2.** Example domain class model for the production material quality control (class attributes and operations are omitted)

Data flow elements (data object, output/input PIN) in the process model can be linked with domain class elements. For example, the type specification attribute for the `qc_result` data flow is set to the class `QCResult` (Fig. 4). This model element connection is later analyzed and used for the transformation, e.g. class operations with the appropriate parameters can be generated.

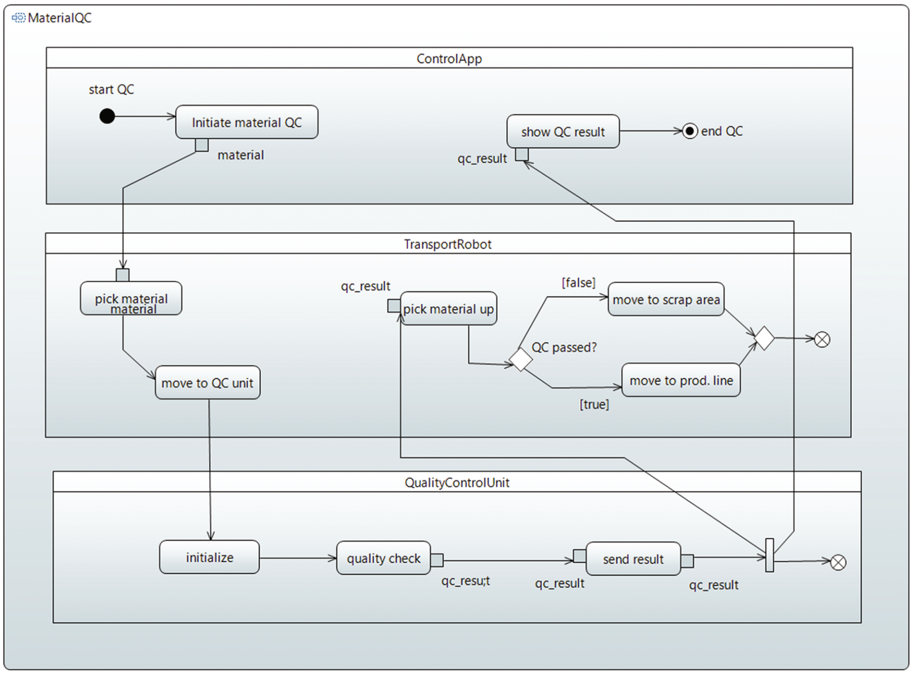
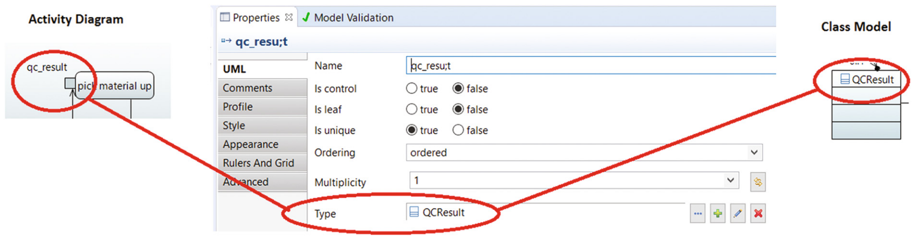


**Fig. 3.** Example process model for the quality control (simplified UML activity diagram)



qc_resu;t

Name: qc_resu;t
Is control: true [✓ false]
Is leaf: true [✓ false]
Is unique: [✓ true] false
Ordering: ordered
Multiplicity: 1
Type: QCResult

**Fig. 4.** Input PIN in the activity diagram with a connection to the domain class `QCResult`

This "weaving" of model elements across different model types is not only important for model transformations, but also for model validation (consistency checking). The following MOM Profile also uses this technique for the combination of behavioral and structural model elements as a preparatory step for the creation of design models.

## 2.3 MOM Profile Definition and Application

The MOM Profile is used for the UML activity diagrams and class models. It provides Stereotypes for mainly two different architectural components: Normal application software and MOM components. This differentiation is important for the M2M transformation. A detail of the MOM Profile is shown in Fig. 5. Process elements have a reference to the structural definition (class, package, or component).

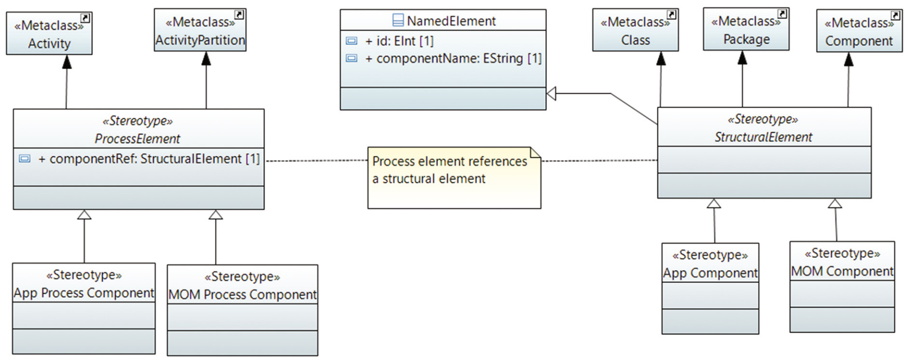

«Metaclass» Activity
«Metaclass» ActivityPartition
NamedElement
+ id: EInt [1]
+ componentName: EString [1]
«Metaclass» Class
«Metaclass» Package
«Metaclass» Component
«Stereotype» ProcessElement
+ componentRef: StructuralElement [1]
Process element references a structural element
«Stereotype» StructuralElement
«Stereotype» App Process Component
«Stereotype» MOM Process Component
«Stereotype» App Component
«Stereotype» MOM Component

**Fig. 5.** Detail of the MOM Profile (simplified model)

Since partitions in an activity diagram can be interpreted as a structural element and used for M2M transformations, it seems questionable why an additional reference to an element of a structural model (class model, component diagram) is necessary. The reason is that an activity is a composite for a partition (that inherits from `ActivityGroup`). Therefore, a Partition cannot be “reused” in other activities (Fig. 6).

The application of the MOM Profile gives software architects the possibility to model architectural aspects while creating or modifying the class model: For the control app, a new package is introduced and – like the classes `RobotControl` and `QCControl` – marked with the Stereotype «App Component». The class `QCOntrol` is created and marked as a «MOM Component» (Fig. 7).

Also, the activity diagram is modified by applying the MOM Profile. Figure 8 shows the new partition for the MOM that acts as a message broker between the different system units, i.e. `ControlApp`, `TransportRobot`, and `QCControl` communicate with each other via the MOM `MaterialQCService`.

The last step before M2M transformations can take place is the creation of references between partitions in activity diagrams and classes or packages in the class model, so that different activity diagrams (partitions) can reference the same app or MOM component (structural elements). Figure 9 depicts the creation of this connection (or weaving) with the attribute (tagged value) of the Stereotype (the partition is on the left side, the package is on the right side, and the `componentRef` is in the middle).

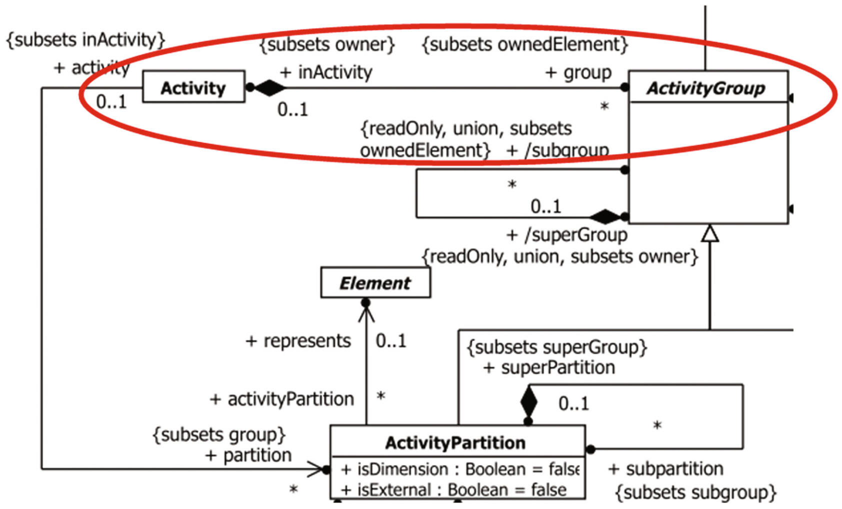


**Fig. 6.** Detail from the UML meta-model for activity diagrams [16, p. 404]

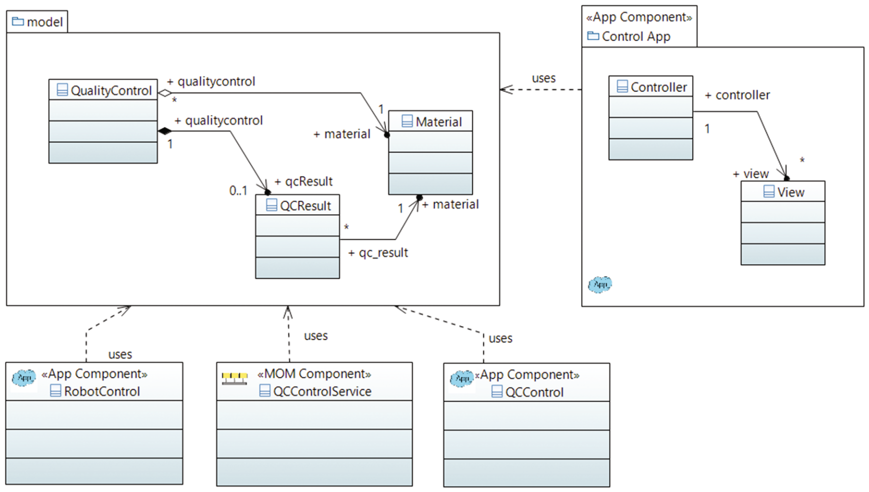


**Fig. 7.** Architectural class diagram with dependencies

## 2.4 Model-to-Model Transformation

The model-to-model transformation mapping needs to be specified before it can be implemented as an operational QVT transformation with QVTo. Rules for the mapping are formulated, e.g. a class in the source model marked with the Stereotype «App Component» will be transformed into a new controller and view class (Table 1).

Mapping functions are defined in operational QVT for classes that are marked with the Stereotype «MOM Component»: A package and a class with a consumer operation

**Table 1.** Detail of the M2M transformation mapping specification

| Source model element | Target model element(s) | Remark |
|---|---|---|
| «MOM Component» class | (a) «MOM Component» classes (b) Package for MOM comp. | New package and classes for the component are created |
| «App Component» class | (a) Controller and view classes (b) Package for app comp. | The MVC pattern is used |
| Action (MOM partition) | Operations for MOM or app classes | References to the MOM classes/packages are used |
| Action (app partition) | Operations for controller class | References to the app classes/packages are used |

are created for each MOM component (Fig. 10). The execution of M2M transformations lead to several software design models (PSM from the viewpoint of the domain model): For every component (app, MOM), a separate package is generated: App components are transformed into packages with classes for the view and controller (MVC) and MOM components are transformed to new packages with a handler for each communication channel (Fig. 8).

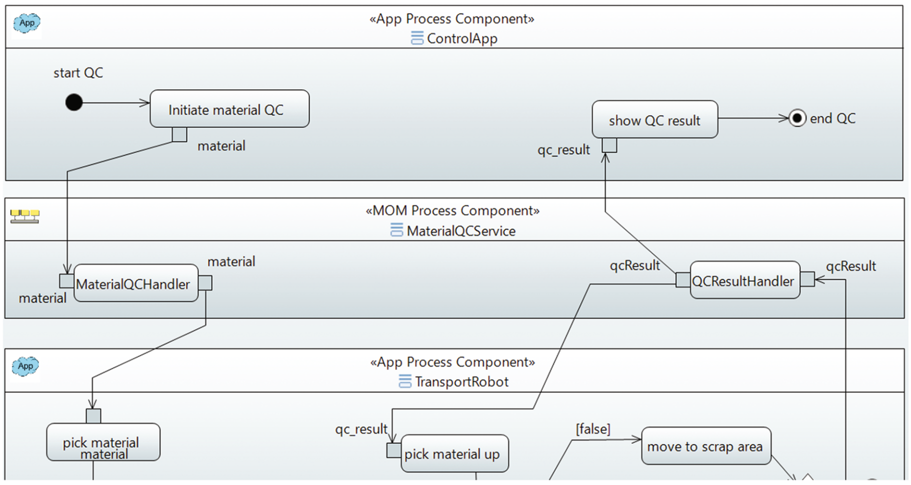
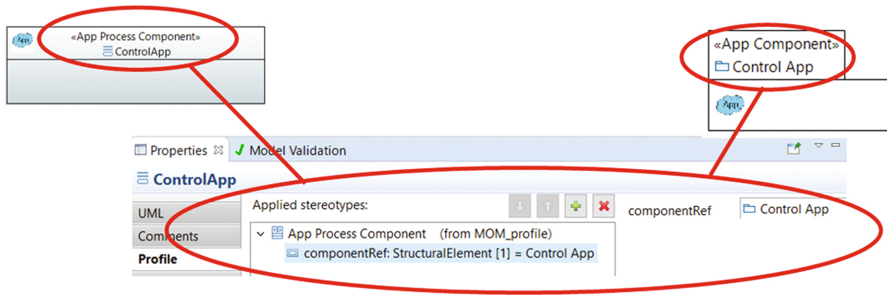


**Fig. 8.** Marked activity diagram with app and MOM process components



Applied stereotypes:
App Process Component (from MOM_profile)
componentRef: StructuralElement [1] = Control App
componentRef: Control App

**Fig. 9.** Connection between marked partition and package

```
mapping Class :: transformMOMClass2ServicePackage() : Package
   when { not self.name.oclIsUndefined() and self.isStereotypeApplied(momComponentStereotype)}
{
    init { // initialization: set names for population
        var packageName := self.name + "Service";
        var qualifiedName := self.qualifiedName;
        var namesSpace := self.namespace;
    }
   population {
        object result:Package {
            name := packageName; // set the package name
            packagedElement += self -> map createMessageHandler(); // create service class
            packagedElement += self -> map createRouteBuilder(); // create route builder
            result.applyStereotype(momComponentStereotype); // apply MOM component to result package
```

**Fig. 10.** QVTo transformation specification for MOM classes (detail)

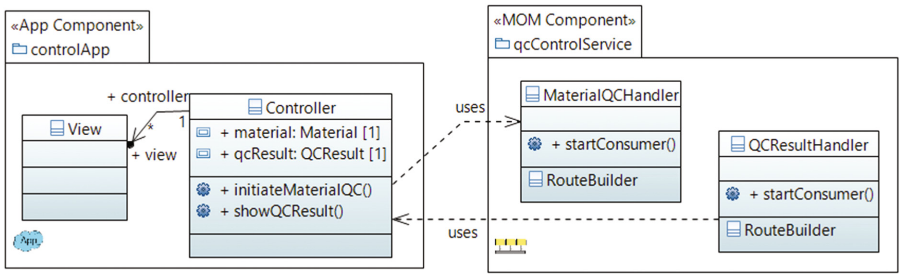


**Fig. 11.** Model-to-model transformation: generated class model

The following class model is a detail of the result of the M2M transformation (Fig. 11). This PSM represents a first version of a software design model.

## 2.5 EIP Profile Application and Code Generation

The software architect can manually modify design models concerning implementation aspects. In this case, implementation-oriented design decisions for the MOM service

**Table 2.** Icons and Stereotypes for the EIP Profiles

| Icon [19] | Stereotype | Remark |
|---|---|---|
| EP | «Endpoint» | An endpoint is a component or an application |
| → | «Channel» | A channel connects two or more endpoints |
| ● | «Message» | |
| ⇉ | «Publish-Subscribe Channel» | Decouples producer and consumer. The publisher broadcasts a message to all subscribers |

package are demonstrated: The cyclic dependency between the package `controlApp` and `QCControlService` will be eliminated (Fig. 11) by applying the enterprise integration pattern (EIP) Publish-Subscriber-Pattern which is part of the EIP Profile for UML class diagrams (Table 2).

The class `QCResultHandler` is marked with the Stereotype «Publish-Subscribe Channel» and the controller (package `controlApp`) becomes a subscriber. This leads to a unidirectional dependency of the package `controlApp` from the MOM service component in the package `qcControlService` (Fig. 12).

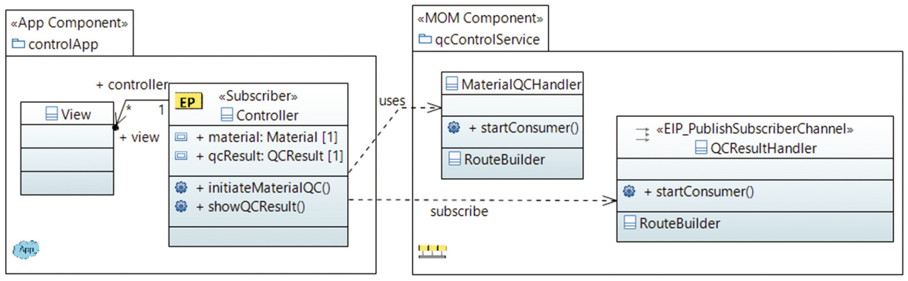
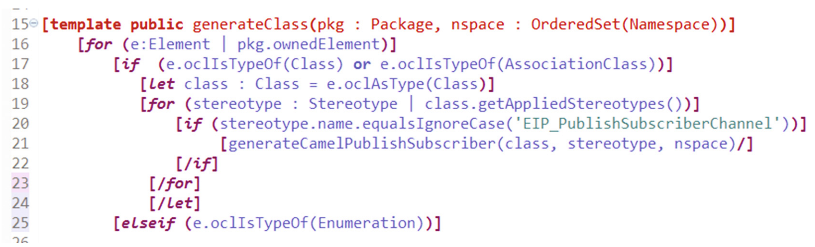


**Fig. 12.** Class diagram (detail) with applied "Publish-Subscriber Channel" EIP



```
[template public generateClass(pkg : Package, nspace : OrderedSet(Namespace))]
    [for (e:Element | pkg.ownedElement)]
        [if  (e.oclIsTypeOf(Class) or e.oclIsTypeOf(AssociationClass))]
           [let class : Class = e.oclAsType(Class)]
           [for (stereotype : Stereotype | class.getAppliedStereotypes())]
               [if (stereotype.name.equalsIgnoreCase('EIP_PublishSubscriberChannel'))]
                    [generateCamelPublishSubscriber(class, stereotype, nspace)/]
               [/if]
            [/for]
            [/let]
        [elseif (e.oclIsTypeOf(Enumeration))]
```

**Fig. 13.** Acceleo template for the code generation of Apache Camel routes (detail)

When the class model is ready for code generation (M2T transformation), a code generator for the app and MOM components is used: With the MOFM2T [20] implementation Acceleo, code generation templates for the PSI were developed. Figure 13 shows a detail of a template for the target platform (API) Apache Camel [21].

## 3 Conclusion

The model-driven approach for MOM development presented in this paper is a first proof-of-concept. The method consists of the following steps:

- Domain Modeling, e.g. creation of UML class model and activity diagram (PIM)
- MOM Profile application for domain model elements (PIM markup)
- Model-to-Model transformation (PIM to PSM): creation of software design models
- Design model modification and EIP Profile application for code generation
- Model-to-Text transformation (code generation: PSM to PSI)

The approach has proven useful, because most of the code for the MOM layer can be generated and it allows manual modifications of software design models. M2M reduces the complexity of the transformation specifications for the code generation: The UML (Profiles, activity diagrams, and class models) have been successfully used. In the future, more EIP patterns and model types will be included in the Profile.

## References

1. Meulen, R. (Gartner): Gartner says 6.4 billion connected “things” will be in use in 2016. www.gartner.com/newsroom/id/3165317. Accessed 1 Feb 2017
2. Unger, H., Meesad, P., Boonkrong, S.: Recent advances in information and communication technology. In: Advances in Intelligent Systems and Computing, vol. 361. Springer, Heidelberg (2015)
3. Bergner, S., et al.: Networked IT-Security for Critical Infrastructures – The Research Agenda Of VeSiKi. www.itskritis.de. Accessed 14 Jan 2017
4. Slama, D., Puhlmann, F., Morrish, J., Bhatnagar, R.M.: Enterprise IoT: Strategies and Best Practices for Connected Products and Services. O’Reilly Media (2015)
5. Organization for the Advancement of Structured Information Standards (OASIS): OASIS Advanced Message Queueing Protocol (AMQP) v1.0, OASIS Standard (2012)
6. International Organization for Standardization: ISO/IEC 19464:2014 - Information technology - Advanced Message Queuing Protocol (AMQP) v1.0 specification (2014)
7. Java Community Process (JCP): Java Message Service (JMS) API. Final Release 1.1 (2002)
8. Organization for the Advancement of Structured Information Standards (OASIS): MQTT Version 3.1.1 Plus Errata 01 (2015)
9. Consel, C., Kabac, M.: Internet of Things: a challenge for software engineering. In: ERCIM - The European Research Consortium for Informatics and Mathematics, Special Theme. www.ercim-news.ercim.eu. Accessed 23 Jan 2017
10. Mukhopadhyay, S.C. (ed.): Internet of Things: Challenges and Opportunities. Springer: Smart Sensors, Measurement and Instrumentation 9 (2014)

11. Holler, J., Tsiatsis, V.: From Machine-to-Machine to the Internet of Things: Introduction to a New Age of Intelligence, 1st edn. Academic Press, London (2014)
12. Hohpe, G., Woolf, B.: Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions, 1st edn. Addison-Wesley, London (2003)
13. Buckl, C., Sommer, S., Scholz, A., Knoll, A., Kemper, A.: Generating a tailored middleware for wireless sensor network applications. In: IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing (sutc 2008)
14. Plšek, A., et al.: A component framework for java-based real-time embedded systems. In: Middleware 2008, ACM/IFIP/USENIX 9th International Middleware Conference (2008)
15. Object Management Group (OMG): Model Driven Architecture (MDA) - MDA Guide, rev. 2.0, document number ormsc/14-06-01 (2014)
16. Object Management Group (OMG): OMG Unified Modeling Language (OMG UML), version 2.5, document formal/2015-03-01 (2015)
17. Object Management Group (OMG): OMG Meta Object Facility (MOF) Core Specification. Version 2.5.1, document number formal/2016-11-01(2016)
18. The Eclipse Foundation: Eclipse Modeling project (2017). www.eclipse.org/modeling Accessed 3 Feb 2017
19. Hohpe, G.: www.enterpriseintegrationpatterns.com. Accessed 21 Jan 2017
20. OMG (Object Management Group): MOF model to text transformation language (MOFM2T). Version 1.0, document number formal/2008-01-16 (2008)
21. Apache Software Foundation: Apache Camel: Publish Subscribe Channel (2017). www.camel.apache.org/publish-subscribe-channel.html. Accessed 1 Feb 2017