Combining Aspect-Oriented and Model-Driven Development for Supply Network Monitoring

Alexander Hornung, Jörg P. Müller
Department of Informatics
Clausthal University of Technology
Julius-Albert-Str. 4
38678 Clausthal-Zellerfeld, Germany
{alexander.hornung, joerg.mueller}@tu-clausthal.de

Abstract. The dissemination of sensor network infrastructures as well as the application of positioning and automatic identification technologies enables a comprehensive monitoring of business processes in supply networks. However, a method which allows specifying monitoring requirements related to business processes in a computation-independent manner is still missing. We propose a method based on model-driven software development to model and implement supply network monitoring applications. To specify event aggregation requirements we enhance a model of a business process by applying model modification techniques in order integrate models of monitoring aspects. The integration of monitoring aspects into a process model is illustrated by an example.

Keywords. Model-driven software development, aspect-oriented modeling, supply network monitoring

1. Introduction

In his vision of ubiquitous computing [15] Mark Weiser describes sensor-equipped computing devices seamlessly integrated into the environment providing useful services. The realization of the ubiquitous computing vision enables an end-to-end monitoring of business processes in supply networks. To be able to implement supply network monitoring (SNM) applications it is required to integrate numerous sensors (event publisher) into a software system. To detect business-relevant events and thus filter the flood of events it is necessary to specify appropriate event aggregation processes. We present a model-driven software development process for SNM applications. We describe in detail the reusable and maintainable encapsulation of aspect models for event aggregation in SNM.

The paper is structured as follows. Section 2 summarizes the theoretical background of our work. In Section 3 we present the design of a model-driven software development process for SNM applications. Section 4 describes the combination of aspect models for event aggregation and a model of a business process. Section 5 contains discussions and related work. In Section 6 we conclude the work and discuss areas of future research.

2. Background

2.1 Supply network monitoring

SNM refers to the monitoring of the state of physical objects transformed by business processes [14]. Furthermore SNM is a subset of supply network event management (SNEM). SNEM is about detecting and correcting disruptive events and malfunctions in material transforming business processes [16]. The task of detecting disruptive events based on measurements obtained during process execution is solved by SNM. Examples of disruptive events in supply networks are out-of-stock situations, machine breakdowns, a breakdown of an air conditioning system or the delay of transportation functions or manufacturing functions. Additionally an SNM application is able to detect regular events like the arrival of products at a products receipt. The goals of SNM are the visibility of work in process and stock amounts using localization information of products, more reliable material flows by monitoring environmental conditions and reduced exception handling costs by quicker detection of disruptive events. The visibility of stock amounts in a supply network removes one of the causes of the well-known bullwhip effect.

2.2 Aspect-oriented programming

We exploit ideas of aspect-oriented programming (AOP) on the instrumentation of business process

models. Programming with aspects enables the encapsulation of cross-cutting concerns [7]. A cross-cutting concern like logging is functionality which cross cuts the main control flow of an application. AOP defines several language constructs. A join point is a well-defined point within program execution. It is selectable by a point cut. A point cut results in a set of selected join points. The point cut is basically a selection expression to determine where the application code has to be instrumented by advices. An advice is the implementation of an aspect using a point cut. An aspect is a reusable module consisting of advices and point cuts implementing a cross-cutting concern. We shall discuss computation independent models of aspects on event aggregation for SNM in Section 4.1. AOP deals with models of aspects and applications at code level. It avoids code duplication and increases maintainability and clarity. AOP also makes design decisions concerning cross-cutting concerns explicit, like every result of a method should be logged first before returning.

2.3. Model-driven software development

Model-driven software development (MDSD) takes advantage of formal models in order to support the implementation and documentation of software systems. A model is the description of (parts of) a system [8]. A well-defined language is applied to create models. A metamodel is a model of the language describing the elements of the language. Models contain instances of elements of the metamodel. A metamodel itself conforms to a meta-metamodel which is self describing. The Fig. 1 summarizes the main ideas of MDSD.

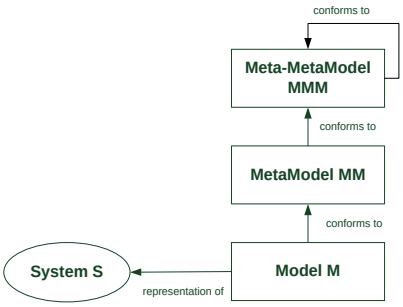

Figure 1. Model-driven software development [2]

The model-driven architecture (MDA) is a framework for MDSD provided by the OMG [8]. The MDA defines three levels of abstraction containing the models which describes a software system. The level of the computation independent model (CIM) describes business-process-related requirements. The platform independent model (PIM) contains a model of a software system implementing the CIM-level requirements. The PIM does not depend on a certain implementation technology. One or more platform specific models (PSM) are generated using a PIM. A PSM describes the software system in a technology-specific manner. The code of the software is generated by a tool which transforms a PSM to code. Applying MDA has several advantages. The productivity of developers will enhance due to technology-independent programming on the PIM level. Portability increases by using a PIM and several transformation definitions to transform to different PSM. Also interoperability between systems will improve using a common PIM for converting between the entities of different systems created using different PSMs.

A model transformation is a function which accepts one or more source models and results in one or more target models [8]. Model transformations are realized by a transformation engine. The transformation engine is controlled by a transformation definition. A transformation definition consists of a set of transformation rules. A transformation rule defines a mapping between elements of the source model's metamodel elements and the target model's metamodel elements. A model modification is another model operator. It manipulates elements of a given model. Also new model elements (instances) are created by a model modification.

3. Development process overview

3.1. Abstraction levels and model types

Table 1 illustrates the MDA-related abstraction levels and the proposed model types.

Table 1. Model types

Abstraction level	Model types
CIM	models of event aggregation for supply network monitoring and a model of the business process to monitor, both modeled as extended event driven process chains
PIM	a model of a supply network monitoring application in conformance to a metamodel for event-based software architectures
PSM	PSM of a supply network monitoring application, e.g. based on a CORBA event service metamodel or java message service metamodel

Section 4 contains explanations concerning the CIM for SNM applications. In the following Section 3.2 we sketch the design of a metamodel for the event-based software architecture which will be used as a language for PIMs. Section 3.3 contains the general steps necessary for modeling, implementing and

deploying SNM applications. We do not describe a PSM-level metamodel in that paper because this is a subject of future research.

3.2. A PIM-level metamodel for supply network monitoring applications

In order to obtain the advantages of the MDA we model a metamodel for PIMs of SNM applications. The metamodel is presented in Fig. 2. The PIM is restricted to the event-based software architecture. Software architecture defines the structure of a software system in terms of system components and relationships among system components. The event-based software architecture constitutes of system components communicating business-process-relevant events utilizing a message-based communication infrastructure [9].

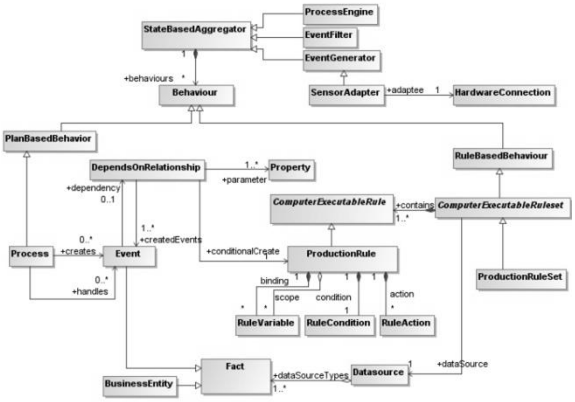

Figure 2. PIM-level metamodel for event-based architecture

During business process execution potentially many events will be generated. An event is a detectable condition that can trigger a notification [5]. A complex event is an event created by aggregation on other events. The aggregation is performed utilizing spatial and time-based conditions. A notification is a signal which is triggered by an event. The recipients of notifications are determined at runtime.

The subscription is the process of linking event publisher with event subscriber. Subscriber and publisher are the two roles of an event-based software system. An event publisher is able to detect and publish events. An event subscriber consumes events according to event filter conditions specified during subscription. Applying direct subscription a subscriber subscribes directly to event publishers. Event publishers are responsible for event generation and optionally for event filtering. In the case of indirect subscription there is a third component which takes the role of the event filter.

We model SNM applications at PIM-level as event-based software systems with indirect event subscription. The intended software architecture comprises of the component types: event generator, event filter and process engine. The event generator is able to measure attributes of its physical environment. For this purpose it is able control sensor hardware. An event generator executes plans for preprocessing and correcting measurements. Furthermore he is optionally able to filter events. An event generator takes the role of an event publisher only. An event filter is at the same time event publisher and event subscriber. The event filter is a subscriber for events in event generators and publishes events to subscribed process engines. He is able to filter events and create complex events using filter rules. The behavior of an event filter type is modeled by an instance of RuleBasedBehavior. The ComputerExecutableRuleset is a RulebasedBehavior.

The ComputerExecutableRuleset is a metaclass of the production rule representation metamodel [10]. A production rule specifies the conditional execution of one or more actions. A RuleAction instance is for example the detection and creation of a complex event. The production rules work on a set of facts of certain types. This set of facts should be able to represent the state of the environment during business process execution. A process engine is able to enact several information transforming business process instances. The behavior of a process engine can be described by several types of business processes (instances of the Process metaclass). We can use the pim4soa process view (cf. [1]) to further detail the Process metaclass.

A business process (process metaclass) is triggered by events and may generate events of certain types. We can conclude that only the detection of business process related events is relevant in SNM applications. Also graphical client applications which display the execution state of business process instances are only interested in business process related events. This leads to the following consequences in modeling the requirements of event aggregation performed by the aforementioned event filters.

Event aggregations for SNM should be modeled by domain experts and modeling experts because they are in most cases able to describe appropriate event types. Thus a well-known modeling language should be applied for event aggregation specification.

Models of business processes instrumented with event aggregation should be readable and maintainable by domain experts and modeling experts.

Additionally, models of event aggregation may be reusable for the monitoring of other business processes.

3.3. Overall development process for supply network monitoring applications

The process which is necessary to model, implement and deploy a SNM application consists of the following steps:

1. Create a model of the business process to monitor.
2. Combine the business process model with a set of predefined reusable models of event aggregations for SNM on CIM level.
3. Create a data model for event aggregation on PIM level containing entity types (instances of the BusinessEntity metaclass) to represent the execution environment of a business process.
4. Transform the instrumented business process model to instances of ProductionRule and Event in order to specify the behavior of an event filter type for the given type of business process.
5. Refine the production rules by adding additional fine-grained events and rules accessing facts on the data model.
6. Model instances of event generators and process engines on PIM level.
7. Transform the PIM to a PSM and finally to source code.
8. Instantiate and initialize the software components necessary for the monitoring of an instance of the business process modeled in step 1.
9. Execute the subscription process to wire all software components.

The remainder of this paper reports on ongoing work realizing the presented development process. In particular the implementation of the steps 1 and 2 are described in Section 4.

4. Combining models of monitoring aspects and a business process model

The steps 1 and 2 of the development process presented in Section 3.3 will be illustrated by an example. Section 4.1 describes a simplified cross-enterprise business process and three aspect models of event aggregations for SNM. The implementation of the combination of the business process model and several aspect models by model modification is explained in Section 4.2. The aspect models and the business process model are created using the notation of extended event-driven process chains (EPC). We refer to [11] and [12] for further information on EPCs.

4.1. Business process model and aspect models

A specification of business-process-related events is created by combining a model of the business process under consideration with aspect models containing event aggregations for SNM. The elements of the aspect models contained in the resulting model of that combination are very similar but are treated as definition copies i.e. as different objects. We use the example of the business process given in Fig. 3 to illustrate the model modifications necessary to combine aspect models with a business process model. To ensure reusability, maintainability and clarity monitoring aspects are modeled in separation to the business process model.

The example in Fig. 3 describes a cross-enterprise business process between a supplier for central processing units (CPU supplier, left swim lane) and a computer manufacturer (right swim lane). Other business processes like the delivery of food may have a similar structure. Note that this business process contains information flows as well as material flows. For example, if shipping units are packed, they will be transported to the products issue of the CPU supplier. Also the packed shipping units are the precondition for posting a dispatch notification to the computer manufacturer. The dispatch notification contains the delivery date, the estimated lead time of delivery and the identifications on all products contained in the shipment units of one delivery. Information-transforming functions like “Register products issue” have to be implemented by sub processes running on process engines. Neither the aspect models nor the model of the business process contain control flows for handling failure events. Also the reaction on regular events is only modeled by a function. Further modeling of reactions on event types will be contained in behavioral models for process engines and the business processes they enact.

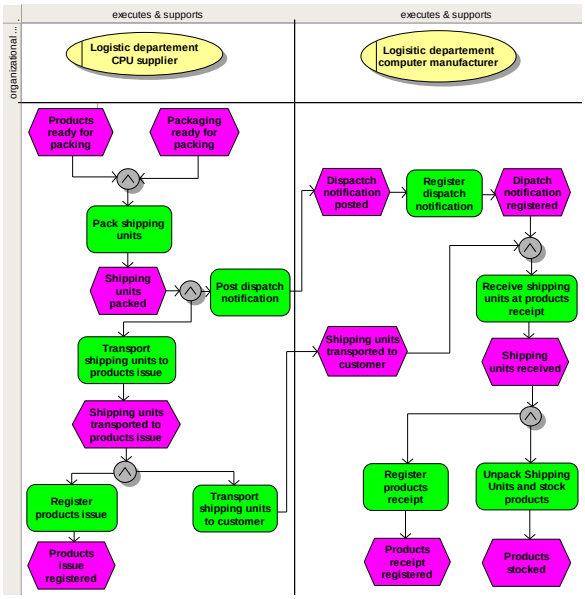

Figure 3. Cross-enterprise business process

Functions of the presented business process are monitored by other functions contained in aspect models of monitoring concerns. Table 2 shows the assignment relationships between aspect models for SNM and functions of the business process to be monitored. In the following, we present aspect models used by this example.

Table 2. Assignment of aspect models

Function	Aspect-models to assign
Transport shipping units to products issue	Shipping unit localization aspect model
Receive shipping units at products receipt	Shipping unit localization aspect model
Transport shipping units to customer	Temperature control aspect model, delay detection aspect model
Unpack shipping units and stock products	Temperature control aspect model

The aspect model in Fig. 4 describes the fact that the temperature in the execution environment of a function (placeholder Function#) is monitored in parallel to the execution of that function. If the increase of the temperature value is higher than a predefined threshold, a temperature failure event will be raised. With the help of this aspect model one can also specify aspect models to monitor other continuous attribute types.

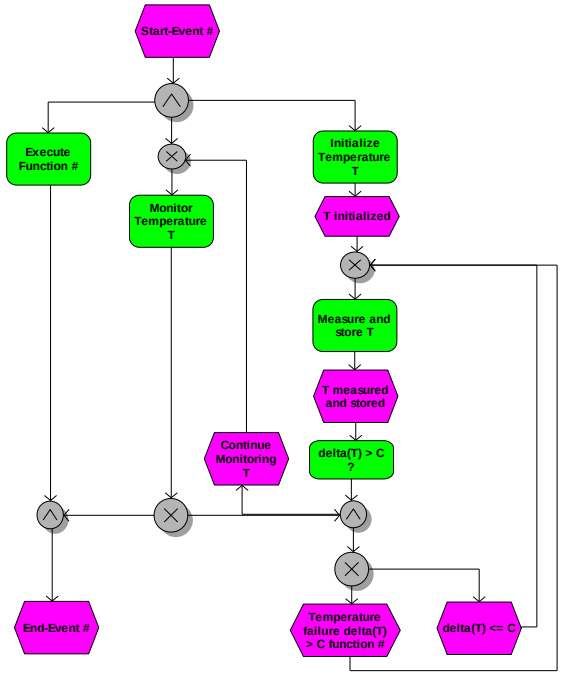

Figure 4. Temperature control aspect model

The delay detection aspect model is presented in Fig. 5. In order to detect the delay of a monitored function (placeholder MonitoredFunction#) a timer is set. The time is set according to the start date and the estimated lead time of the monitored function. This information has to be accessible during the execution of the function "Determine start date D and lead time ld of the monitored function". If the timer expires before the end of the monitored function a delay event is generated. The post condition for functions transporting shipping units is monitored by constructs contained in the shipping unit localization aspect model of Fig. 6. The aspect model describes that a function can be treated as finished only if all shipping units of that function can be localized at a certain location (placeholder place#). If the localization fails for a predefined number of localization cycles a localization failure event is created.

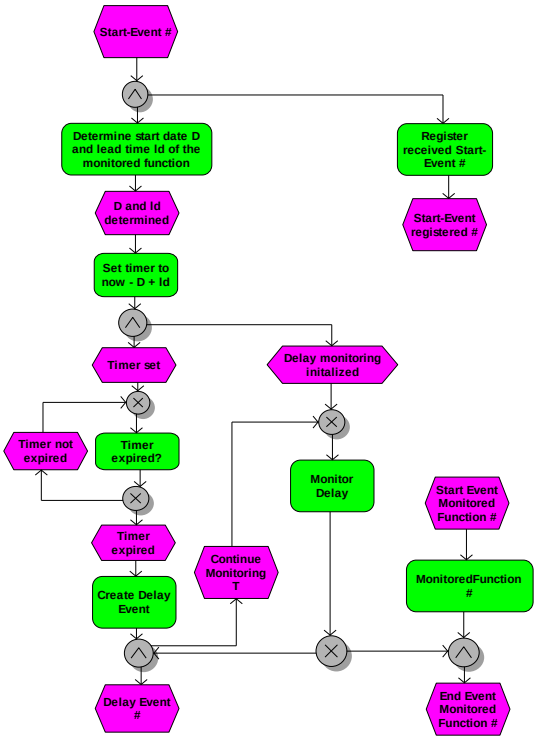

Figure 5. Delay detection aspect model

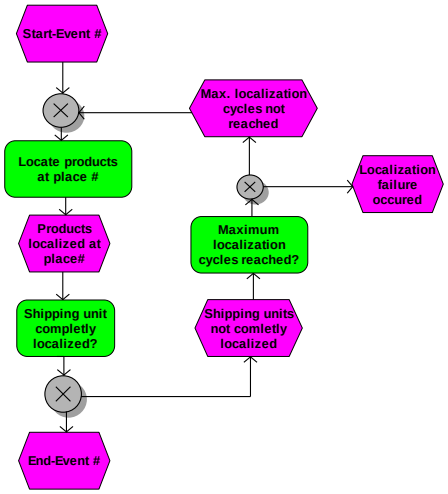

Figure 6. Shipping unit localization aspect model

4.2. Implementation approaches for model modification

Having described all aspect models and the business process to monitor we are now ready for describing the combination of the various models which is called

weaving in AOP terms. By model modification it is possible to combine aspect models and a business process model. We apply that combination of models on CIM level.

The model modification is performed using openArchitectureWare (OAW) [4]. OAW provides the Xtend language for model modification and model transformation. Xtend is an object constraint language (OCL)-based language. Unlike OCL Xtend supports the creation and manipulation of model elements. In order to create EPC models for model modification by OAW and Xtend the EPC metamodel was implemented using the ecore eclipse modeling framework (ecore EMF) meta-metamodel [13]. In the following we discuss three approaches for combining aspect models and a business process model.

(1) Aspect models contained in the model modification source code and generated during model modification,
(2) Aspect models and model modification in separate and
(3) Applying aspect models, a model weaver and post processing operations.

4.2.1 Aspect models generated during model modification

This type of model modification contains the instructions for constructing aspect models in the source code of the model modification. For example to create model elements (instances) contained in a localization aspect model the `localizationProcess` operation has to be invoked during model modification. The `localizationProcess` operation is shown in Fig. 7.

```
localizationProcess(Process process, String param, Integer
occId):
      createFunctions(process, param, occId) ->
      createEvents(process, param, occId) ->
      createOperators(process, occId) ->
      createEdges(process, occId)
;
createFunctions(Process process, String param, Integer occId):
      process.processElems.add(
      createFunction("Locate product at " + param ,
      "LocateProduct", occId)) ->
      … ;
create Function createFunction(String name, String id, Integer
occId):
      this.setName(name) ->
      this.setIdentifier(id) ->
      this.setOccurrenceId(occId);
...
```

Figure 7. Operation localizationProcess

The model modification process which instruments the given business process model with the entire set of aspect models works as follows.

1. Read the business process model from a file and instantiate all model elements.
2. Call the first model modification operation which creates elements of the temperature control aspect model for each function a temperature failure has to be monitored.
3. Call the second model modification operation which creates elements of the delay detection aspect model for each function to monitor a possible delay.
4. Call the third model modification operation which creates elements of the location aspect model for each function to monitor. This operation will replace the function to monitor by the event aggregation for shipping unit localization.
5. Serialize and store the modified business process model.

4.2.2 Separation of aspect models and model-modification

This type of model modification separates model modification code and aspect models. As well as the EPC model of the business process the EPC models of the monitoring aspects are serialized and persistently stored. The Fig. 8 shows the shipping unit localization aspect model in an EMF-based editor for EPC models.

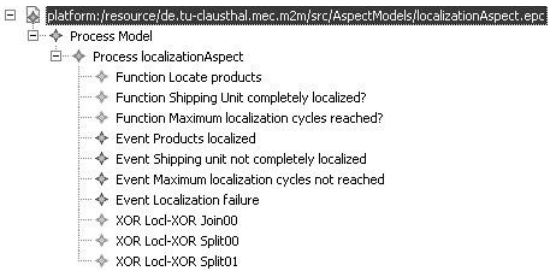

Figure 8. Shipping unit localization aspect model

For each aspect model we invoke the operation `add<AspectName>Aspect` during model modification. The Fig. 9 depicts the `addLocationAspect` operation. The functions "Transport shipping units to products issue" and "Receive shipping units at products supply" are each instrumented by a shipping unit localization aspect model. Instrumentation is performed by the overloaded `weaveElemsLocationMonitoringElements` operation. The input parameters of that operation are model elements which connects the business process model with an aspect model. These parameters are similar to point cuts in AOP.

```
addLocationAspect(ProcessModel baseProcessModel, ProcessModel
aspectProcessModel):

      baseProcessModel.addAspectModel(aspectProcessModel,0) ->
      baseProcessModel.addAspectModel(aspectProcessModel,1) ->
      baseProcessModel.process.weaveElemsLocationMonitoringElements();

private Void weaveElemsLocationMonitoringElements(Process process):

process.weaveElemsLocationMonitoringElements(
      process.processElems.select(e|e.identifier == "AndSplit00").get(0),
      process.processElems.select(e|e.identifier ==
      "TransportShippingUnitsToProductsIssue").get(0),
      process.processElems.select(e|e.identifier ==
      "ShippingUnitsTransportedToProductsIssue").get(0),
      1)
->
process.weaveElemsLocationMonitoringElements(
      process.processElems.select(e| e.identifier == "AndJoin01").get(0),
      process.processElems.select(e|e.identifier ==
      "ReceiveShippingUnitAtProductsReceipt").get(0),
      process.processElems.select(e|e.identifier ==
      "ShippingUnitsReceived").get(0),
      0);
```

Figure 9. Operation addLocationAspect

The Fig. 10 contains an excerpt of the overloaded `weaveElemsLocationMonitoringElements` operation. The operation shows that the edges between aspect model elements and the model elements of the business process model have to be created. For example the reference to the monitored function has to be removed from the successor collection of the `startElement` parameter. The value of the `startElement` parameter is the point cut selected in the business process model (which is the original predecessor of the monitored function, e.g. AndSplit00). A new reference has to be inserted into that collection to create an edge to the first element in the aspect model (Locl-XorJoin00).

```
private Void weaveElemsLocationMonitoringElements(
Process process, ProcessElement startElement, ProcessElement
functionToMonitor, ProcessElement endEvent, Integer occurrenceId):


startElement.setSuccessor(createList(
            startElement.successor.select(e|e.identifier !=
            functionToMonitor.identifier)))
->
startElement.setSuccessor(createList(
                  startElement.successor,
                  process.processElems.select(e|e.identifier ==
      "Locl-XorJoin00" && e.occurrenceId == occurrenceId).get(0)))
->
process.processElems.select(e|e.identifier == "Locl-XorJoin00" &&
e.occurrenceId == occurrenceId).setPredecessor(
createList(process.processElems.select(e|e.identifier == "Locl-
      XorJoin00" && e.occurrenceId == occurrenceId).predecessor,
      startElement))
-> …
;
```

Figure 10. Operation weaveElemsLocationMonitoringElements

4.2.3 Aspect models, model weaving and model modification

For this type of model modification we use the Xweave model weaver which is part of OAW. To insert model elements of aspect models into the business process model model-elements from both models must match. Matching works either by defining a point cut expression to select model elements in the base model (the business process model in this case) or by the equality of the names of the model elements. The temperature control aspect model of Fig. 11 states that `processElement` instances from the temperature control aspect model have to be inserted into the `processElems` collection of `Process` instances (Process *) of the base model.

Figure 11. Aspect model applicable for model weaving

Model elements with the same name in both models (e.g. the function "Transport shipping units to customer") are merged into one instance containing all the attribute values from its originating instances.

5. Discussion and related work

We compare the approaches presented in Sections 4.2.1–4.2.3 with respect to maintainability and reusability of the aspect models.

The first approach (Section 4.2.1) negates the opportunities of aspect orientation. By directly encoding the model in the modification code the definition of complex aspect models becomes unclear and poor maintainable in conclusion. Like all other approaches the first approach provides the advantage of an automatically created model of the instrumented business process. This modified model contains a specification of business-relevant event types. The advantages of the second approach (Section 4.2.2) are a clear separation between aspect models and model modification code, the possibility to graphically edit aspect models and the opportunity to reuse aspect models in other business process models. Creating or modifying complex EPC models in a simple EMF-based editor is problematic. The third approach for combing a model of a business process and aspect models (Section 4.2.3) provides the same advantages as the second approach. The disadvantage of applying this approach is that an aspect model is now directly dependent on model elements in a business process model. Additionally the model resulting from model weaving has to be post processed in order to update relationships between model elements of the business process model and the model elements of the added aspect models.

Having implemented three approaches for combining aspect models for SNM and a business process model it points out that the separation of aspect models and the business process model without using a model weaver (the second approach) provides the greatest opportunities for reusability and maintainability of the event aggregation aspect models.

There are also other known approaches combining aspect models and a base model utilizing aspect oriented principles. The authors of [6] describe how to enhance core functionality of a software system by further functionality encapsulated in aspect models. Unlike our approaches [6] shows how to enhance a metamodel for home automation systems using aspects affecting the structure of that metamodel. [3] describes an approach of applying and deriving aspect models in agent-oriented requirements specifications. They use the Tropos formal language in order to create models of requirements on the interaction and on the task structure of plan-based intelligent software agents. The structure of certain aspects is extracted and separated into a new model element. The paper [3] contains algorithms to combine aspect model elements and models of agents. In comparison to this work we focus on modeling of the behavior and the structure of SNM applications. With the help of the development process of Section 3.3 we have explained how the result of weaving a business process model and several aspect models fits into a model-driven software development approach for SNM applications.

6. Conclusion and outlook

The contributions of the paper are threefold. First, we introduced a model-driven development process for SNM applications. Second, a PIM-level metamodel for models of SNM applications restricted to the event-based software architecture has been presented. Requirements on event aggregations have to be modeled with respect to business processes monitored. The implementation and discussion of three approaches for combining a model of a business process to monitor and several aspect models for SNM is the third contribution of that paper. Specifying aspect models for business-process-related event aggregation in SNM provides advantages in reusability, clarity and model maintenance.

There are several areas of future research. First of all we have to investigate algorithms for the transformation from the EPC model of the instrumented business process into production rules specifying the behavior of an event filter type on the PIM-level. To do so it is necessary to recognize certain structures of EPC metamodel elements and transform them into suitable ProductionRule instances of the metamodel proposed in Section 3.2. This transformation will generate coarse-grained event types and aggregation among them. Another area of future research is the transformation from the PIM of a SNM application into a PSM. Evaluation of this transformation should be done by determining an implementation technology (e.g. Java Message Service) and an appropriate PSM-level metamodel. Furthermore we have to demonstrate a complete tool chain for the MDSD process for SNM applications applied to a detailed implementation example.

7. References

[1] Bauer, B., Müller, J., P., Roser, S., A Decentralized Broker Architecture for Collaborative Business Process Modelling and Enactment, Proc. of the 2nd IESA, Bordeaux, France, 2007, pp. 115-126.

[2] Bézivin, J., Barbero, M., On the Applicability Scope of Model Driven Engineering, Proc. of the 4th Int. Workshop on Model-Based Methodologies for Pervasive and Embedded Software, Braga, Portugal, 2007, pp.3 -7.

[3] Chakrawarthy, K., Joshi, R., Capturing Task and Dependency Aspects in Agent Oriented Requirement Specifications, Proc. of the 7th AAMAS, Estoril, Portugal, 2008, pp. 25 - 36.

[4] Efftinge, S., Open Architecture Ware Framework, available at: http://www.openarchitectureware.org, Accessed: 1st June 2008.

[5] Faison, T, Event-Based Programming, Apress, New York, USA, 2006.

[6] Groher, I., Voelter, M., XWeave: Models and Aspects in Concert, Proc. of the 10th Int. workshop on Aspect-oriented modeling, Vancouver, Canada, 2007, pp. 35-40.

[7] Kiczales, G., Irwin, J., Lamping, J., Loingtier, J., M., Lopes, C., Madea, C., Mendhekar, A., Aspect-Oriented Programming, Proceedings of the European Conference on Object-Oriented Programming, Jyväskylä, Finnland, 1997, pp. 220–242.

[8] Kleppe, A, Warmer, J., Bast, W., MDA explained, Addison-Wesley, Boston, USA, 2003.

[9] Michelson, B., M., Event Driven Architecture Overview, Patricia Seybold Group, 2006.

[10] OMG, Production Rule Representation Specification, 2007.

[11] Scheer, A., W., ARIS – Vom Geschäftsprozessmodell zum Anwendungssystem, Springer, Berlin, 2002.

[12] Seel, C., Vanderhaeghen, D., Meta-model based Extensions of the EPC for Inter-Organisational Process Modelling, Proc. of 4. GI-Workshop, Hamburg, Germany, 2005.

[13] Steinberg, D., Budinsky, F., Paternostro, M., Merks, E., EMF – eclipse modeling framework, Addision-Wesley, Boston, USA, 2008.

[14] Strassner, M., Schoch, T., Today's impact of Ubiquitous Computing on Business Processes, Proc. of Pervasive2002, Zürich, Switzerland, 2002, pp. 62-74.

[15] Weiser, M., The Computer for the 21st Century, Scientific American, Vol. 265, No. 3, USA, 1991, pp. 94-104.

[16] Zimmermann, R., Winkler, S., Bodendorf, F., Agent-based Supply Chain Event Management – Concept and Assessment, Proc. of the 39th HICSS, Hawaii, USA, 2006, pp. 1-10.