Weaving Aspects and Business Processes through Model Transformation

Heiko Witteborg[1], Anis Charfi[1], Daniel Colomer Collell[2], and Mira Mezini[2]

[1] SAP AG
Bleichstr. 8, Darmstadt, Germany
{firstname.lastname}@sap.com
[2] Software Technology Group
TU Darmstadt, Hochschulstr. 10, Darmstadt, Germany
lastname@informatik.tu-darmstadt.de

Abstract. Concerns such as logging, auditing and accounting need to be addressed already in the business process modeling phase and not only in the process implementation phase. Mostly, such concerns are modeled as part of the normal flow in business process models. However, the crosscutting nature of such concerns leads to complex, scattered, and tangled models that are hard to understand and to manage. The lack of appropriate means to modularize crosscutting concerns in business process modeling languages seriously affects understandability, maintainability and reusability. In a previous work we proposed AO4BPMN 1.0 as an aspect-oriented extension of BPMN that allows the modularization of crosscutting concerns. However, there were several open issues in that proposal. First, it lacks a concrete weaving mechanism for composing business processes and aspects. Second, it lacks a well-defined pointcut language to select join points. Third, it does not support BPMN 2.0, which was still under development at that time. In this paper we tackle these issues and present a weaver for AO4BPMN based on model transformation as well as an OCL-based pointcut language.

Keywords: aspects, business process, modeling, weaving, model transformation.

1 Introduction

With the rise of executable process languages such as the Web Services Business Process Execution Language (WS-BPEL) [11] and the Business Process Modeling Notation (BPMN) [13] the borders between process modeling and process implementation are blurring. A BPMN process model can be at the same time the process model and the process implementation as BPMN 2 is not only a modeling language but also an executable language. This requires such languages to allow expressing several concerns that are important for process implementation such as security, accounting, and auditing. However, these concerns have a crosscutting nature and using the current means of WS- BPEL and BPMN to specify them leads to complex process models with many activities that are not directly related to the core process logic. Without appropriate language constructs for supporting crosscutting concerns the process complexity increases

M. Villari et al. (Eds.) : ESOCC 2014, LNCS 8745, pp. 47–61, 2014.

and the process models become hard to understand, to manage, and to maintain. For example, consider a business process for order fulfillment that involves the usage of external services with costs such as a delivery service. We need to extend the business process to collect accounting data whenever we use an external service with cost. To do this we need for each activity that calls an external service at least another activity for accounting. Obviously the process complexity will drastically increase and the activities belonging to one concern such as accounting will be mixed with activities addressing other concerns. In addition, the implementation of a given concern such as accounting will be scattered across several business process models.

In addition to the need for concepts to modularize crosscutting concerns in business processes, we motivated in [15] the need for concepts to modularize business process extensions as first-class entities. Consider for example an independent software vendor (ISV) who extends the standard business processes implemented by an ERP-System to adapt that software to the particular needs of a company or a domain. The process extension should be kept separate from the business process for several reasons. First, the extension is owned by the ISV whereas the core process is owned by the ERP provider. A separation of both entities avoids problems when the core process is updated by the ERP provider. Second, several extensions may be applied to the same business process and if they are not modelled as separate entities it would be very difficult to understand the new process that results after applying these extensions to the original core process.

In order to address the limitations of business process languages with respect to crosscutting concern modularity and process extension modularity, we introduced aspect-oriented workflow languages in [3] and as an instantiation thereof we proposed AO4BPEL [4], which is an aspect-oriented extension to WS-BPEL. A new implementation of that language based on Apache ODE was recently made available [9]. After AO4BPEL, we proposed AO4BPMN 1.0 [5] as an aspect-oriented extension to BPMN, which allows a better modularization of crosscutting concerns and process extensions. However, that work has three open issues. First, it lacks a concrete pointcut language. Second, no weaving mechanism was provided. Third, the language extension and the respective editor were based on BPMN 1.2 and not on BPMN 2.0, which was still in development at that time.

In this paper, we present the following contributions. First, we refine the language definition of AO4BPMN and make it compliant with BPMN 2.0 using a light-weight extension of BPMN 2.0. Second, we propose a concrete pointcut language based on OCL. Third, we present a weaving mechanism for composing aspects and processes, which is based on model-to-model transformation. Fourth, we present a new Eclipse based toolset for AO4BPMN 2.0 including a graphical editor, a weaving wizard, and a weaver. A variation of AO4BPMN was applied in an industrial context to modularize business process extensions as first class entities as presented in [15]. However in that work process extensions were extracted from modified business process models and modularized in aspects implicitly behind the scenes. In the current paper, we advocate a rather explicit approach for modeling aspects in business processes.

The remainder of this paper is organized as follows. Section 2 presents AO4BPMN 2.0 and illustrates its constructs by means of an example. Section 3 introduces the weaving

mechanism and the underlying algorithms. Section 4 is dedicated to implementation and tooling. Section 5 discusses related work and Section 6 concludes the paper.

2 AO4BPMN 2.0

In this section, we first give a language overview focusing on the AO4BPMN concepts that were added or refined in version 2.0. Then, we shortly discuss the integration of these concepts into the metamodel of BPMN 2.0. After that, we present an example business process that we use throughout the paper for illustration.

2.1 Language Overview

Base Process and Join Points: In AO4BPMN, a *Base Process* is a business process model that ideally focusses on the core functional process logic, while the crosscutting concerns are captured separately in *Aspect* modules. The base process contains *Join Points*, which are distinct and uniquely identifiable points at which crosscutting concerns can be integrated. AO4BPMN per default defines base processes and joint points in an implicit and standard-conform way: any standard BPMN 2.0 process can serve as a base process. Any contained BPMN flow node, i.e., any BPMN element used to define events, activities and gateways, is available as an AO4BPMN join point. Hence, existing BPMN 2.0 processes are AO4BPMN-enabled by default. A base process modeler does not have to foresee or annotate possible AO4BPMN join points. However, while this gives the modeler flexibility and ensures the standard-conformity and reusability of the process models, there are use cases, in which certain process elements should be protected from being advised by aspects [15]. For example, this requirement arises when a core business process should not be adapted by a third-party to protect intellectual property or to ensure compliance with legal obligations. To cover these scenarios, AO4BPMN also supports the explicit exclusion of base process flow nodes from the join point set.

Aspects: Aspects are units for modularizing crosscutting concerns in business process models such as monitoring, accounting, or security. An aspect acts as a container for both the concern-specific business process logic, represented as *Advice*, and for the *Pointcuts*, which select the join points at which the advice will be integrated. An aspect can define multiple sets of pointcuts and advices. In addition, an aspect may define some state, i.e., data that can be accessed and modified by the advices contained in that aspect.

Advice and Proceed Node: An advice is a construct that holds a connected fragment of the process logic of a crosscutting concern. In case the concern is scattered over several join points in a business process (e.g., a timer that is started at the beginning of a process and is stopped at its end), the corresponding aspect will hold several advices. Apart from the concern's process fragments, an advice may also include the special flow node *Proceed*, which is a placeholder for the *Join Point* element that is selected by the pointcut. This node defines how the advice must be applied: everything implemented

sequentially before the *Proceed* will be integrated before the selected join point; everything implemented sequentially after it will be integrated after the selected join point (cf. Section 3). If the *Proceed* flow node is not used the join point element is replaced by the advice content (or deleted without substitution, if the advice is empty). Compared to other AOP languages, where advice composition strategies are often limited to either before, after, or instead of the join points selected by the associated pointcut, AO4BPMN's *Proceed* concept allows more advanced advice composition strategies.

Pointcut: A pointcut allows selecting one or more related join points at which a certain advice will be integrated. Usually, a pointcut targets elements of a base process. Yet, as multiple aspects can be applied to the same base process, the pointcut belonging to a certain aspect may also target elements that were introduced by another aspect in a previous step, resulting in a dependency hierarchy of concerns. Pointcuts can be defined based on element characteristics both on the meta level (e.g., all activities that are typed as receive task) or on the model level (e.g., an activity with a certain id). The pointcut language should also be powerful enough to allow selecting elements based on different criteria such as their relationship to other model elements (e.g., the resource that will perform an activity), their containment information, or the existence of certain boundary events. To support weaving at execution time, the pointcuts may also include instance attributes, e.g., the state of an activity or the priority of a user task. Based on the alternatives sketched in [5], we opt for an OCL-based pointcut language for AO4BPMN 2.0 as OCL [12] provides the required expressiveness, flexibility and tool support. We leverage OCL's usability and understandability by providing helper functions that facilitate the specification of query expressions, e.g., to enable the use of wild-cards.

2.2 Extending the BPMN 2.0 Metamodel with AO4BPMN Concepts

BPMN 2.0 introduces an extensibility mechanism that can be used to extend BPMN's metamodel in a standard-compliant manner [13]. The concepts *ExtensionDefinition* and *ExtensionAttributeDefinition* are at the core of this mechanism. We use these concepts to define AO4BPMN 2.0 as a light-weight extension of BPMN 2.0. Figure 1 shows using dashed arrows how the AO4BPMN metamodel elements extend to the BPMN metamodel elements. AO4BPMN *BaseProcesses* and *Aspects* are specialized *Processes*. An aspect contains possibly multiple *States* and *Advices*. An *Advice* is a specialized *SubProcess* that contains a *Pointcut* and optionally a *ProceedNode* in addition to the elements a standard *SubProcess* may contain. A *Pointcut* may reference multiple *JoinPoints*, which are specialized *FlowNodes* that are part of the *BaseProcess*.

2.3 Example Scenario: Agile Software Development

We consider the (simplified) scrum-like development process shown in Figure 2 that will be used as base process. This process covers the creation and presentation of a sprint backlog by the development manager, the execution of tasks during the sprint by the development team, and the review of the sprint results. While this development process looks plain and readable, it can easily become complex and tangled by adding

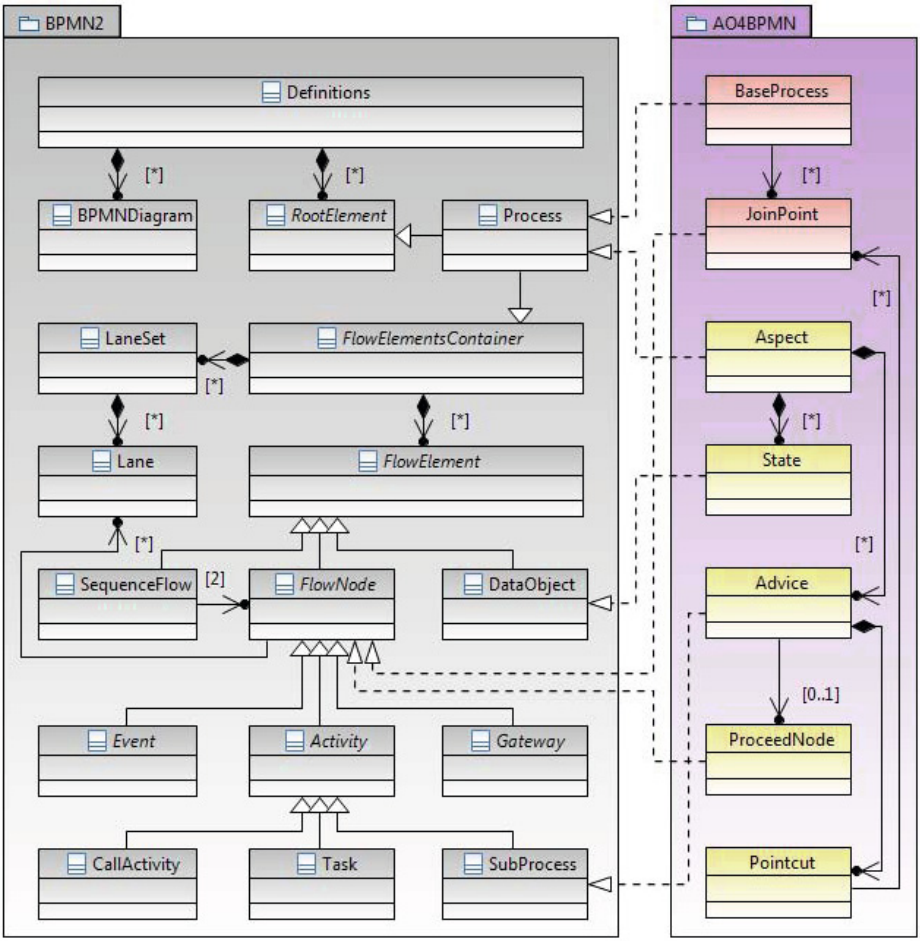

Fig. 1. Extending BPMN 2.0 with AO4BPMN concepts

activities to cover concerns such as quality assurance, test-driven development, monitoring or documentation. Other examples of crosscutting concers could involve activities related continuous integration or setup-specific activities e.g., regarding open-source development; activities related to other development approaches such as extreme programming practices; activities that might be of interest for other company-internal stakeholders like the HR, finance, or legal department. A business process model representing a realistic development setup would have to cover multiple concerns. Mixing the core process functionality and the other relevant crosscutting concerns leads to a complex process model that is difficult to understand and to maintain.

AO4BPMN helps addressing the issues discussed above. Figure 3 shows an AO4BPMN aspect that modularizes the process logic related to test-driven development (TDD) in the context of our example process. For demonstration purposes, two rather simple advices define the additional process fragments required for acceptance

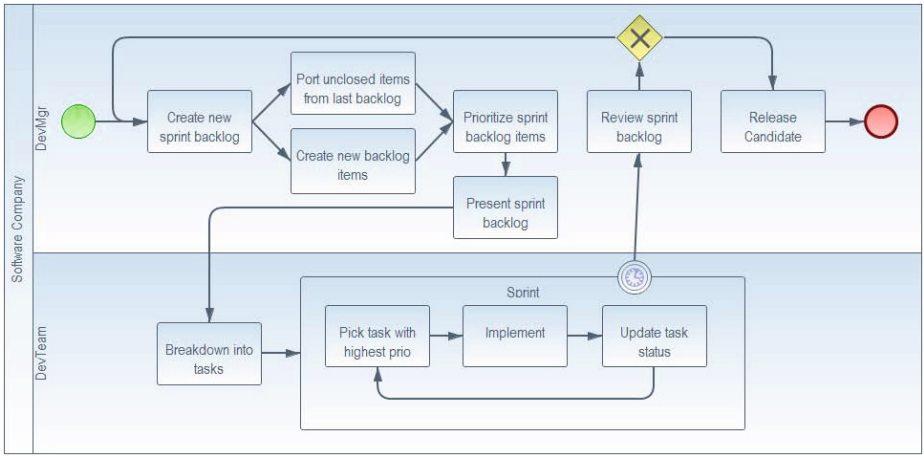

Fig. 2. Agile development base process

tests. The first advice specifies that the definition of acceptance tests should be done before the selected join points (sequentially before the *Proceed* element). The respective pointcut is the OCL expression `context Activity :: name = 'Prioritize sprint backlog items'`. The second advice specifies that running the acceptance tests should be done after the selected join points (after the *Proceed* element). The respective pointcut is `context BoundaryEvent :: name = 'SprintTimeOut'`.

The third advice, *Unit Testing*, shows a more advanced structure, with the *Proceed* element located in the middle. In addition, an end event is used in this advice to indicate that there should be an outgoing sequence flow to the successors of the join point. In analogy to the instantiation semantics of BPMN 2.0, these start and end event elements are not required when the node has no incoming (resp. outgoing) sequence flows. Hence, we do not need to add specialized events in the acceptance test advices. The third advice is intended to refine the implementation phase of our scrum process, thus, the corresponding pointcut is `context Activity :: name = 'Implement'`.

3 Composition of Processes and Aspects

As AO4BPMN aspects are separate modules a weaving mechanism is necessary to compose them with the BPMN base processes. In this section, we present a weaving mechanism that is based on model-to-model transformation together with the underlying algorithms and rules.

3.1 Weaving Mechanism

The weaving mechanism of AO4BPMN takes aspect models and business process models as inputs and produces BPMN 2.0 compliant business process models as output.

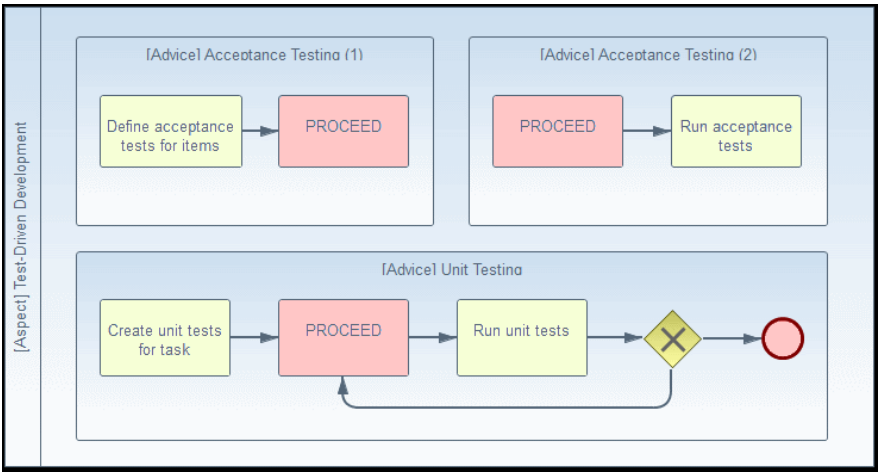

Fig. 3. Test-driven development aspect

Apart from evaluating the pointcut queries and identifying the matching join points, the weaving mechanism integrates the advice process fragments into the base processes via model-to-model transformation. Thereby, merging and branching restrictions need to obeyed and the newly inserted elements need to be re-linked with the base process flow. A simplified version of the weaving algorithm is shown in Listing 1. We elaborate on the core tasks of the weaving mechanism in the following.

```
public List<Process> weave(List<Process> baseProcesses, List<Aspect> aspects){
 List<Process> targetProcesses = new ArrayList<Process>();
 Process targetProcess;
 for (Aspect aspect : aspects) {
   for (Process baseProcess : baseProcesses) {
     targetProcess = cloneProcess(baseProcess);
     for (Advice advice : aspect.getAdvices()){
       List<JoinPoint> jps = identifyJoinPoints(baseProcess, advice.getPointcut());
       for (JoinPoint jp : jps){
         cloneAdvice(targetProcess, advice);
         relink(targetProcess, jp);
       }
     }
     targetProcesses.add(targetProcess);
   }
 }
 return targetProcesses;
}
```

Listing 1. Weaving algorithm

Join point identification phase: The first phase in the weaving process is the evaluation of the pointcuts contained in the aspect. This phase identifies the join points at which the advices have to be integrated in the subsequent phases. Depending on the OCL expressions used in the pointcuts the join point selection can be based on different characteristics of the flow nodes, such as instance-specific properties (e.g., select a node based on its id or name), properties on the meta-level (e.g., select all nodes of type user

task) or context patterns (e.g., select all nodes that have multiple outgoing edges). These query options can be combined, aggregated, or generalized by using the respective OCL constructs.

In our agile software development scenario, the TDD aspect defines instance-specific pointcuts: the acceptance test definition should be done before prioritizing the backlog items; the unit testing should be done around the implementation task; and the acceptance test should happen after the sprint has finished. A more sophisticated advice in this example scenario could be defined to enable role-specific KPI monitoring, with a pointcut selecting manual tasks that were defined to be performed by a certain *resourceRole*, e.g., *Developer*. The resulting pointcut expression in OCL could be: `context Task :: resources->select(r : ResourceRole | r.name = 'Developer')->notEmpty()`.

Cloning phase: In this phase, the flow nodes and connection elements defined in the different advices have to be cloned and merged with the selected join points in the base processes. This is achieved by transforming the process models. This cloning phase is responsible for copying the needed information and process elements from the aspect model to the process model. Given a join point and a *Proceed* element as input, the weaver knowledge is limited to whatever is directly attached to these elements. For this reason, the weaver clones level by level. In this context, a level is defined as the set of all the elements that are at the same distance from the *Proceed* element. The distance between two elements is defined as the number of associations in the shortest path that links these two elements.

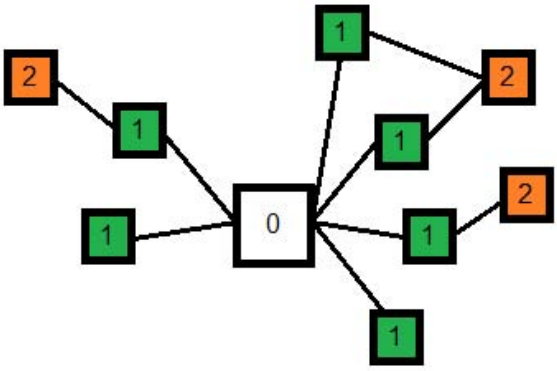

Fig. 4. Abstract structure with the elements and their distance to the central element

For illustration Figure 4 shows an abstract structure with elements and their distance to the central element (element with distance 0). The weaving algorithm clones first all the elements at distance 1, then all elements at distance 2, and so on until the last level is reached. In the unit testing advice of our example, apart from the *Proceed* element itself, all elements have distance 1 except the end event, which has distance 2.

Besides flow nodes, other BPMN 2.0 elements can occur in the advice definition and need to be handled appropriately in the cloning step. Connections are elements of particular interest for the cloning phase. In fact, in a BPMN process diagram the different flow objects and other elements such as data objects are related by means of connecting elements such as sequence flow and association. Moreover, boundary events can be used to interruptively branch the process flow. Regarding the process transformation, these connecting elements are taken into account by the weaver as explained in the following.

- Everything that is linked by means of a sequence flow connection belongs semantically to the process itself as part of its sequence of steps. The weaving algorithm supports this flow level as it is the level that contains the semantic part of the process.
- Everything related by means of an association connection is used to express relevant information that is found along the steps of the process; but it does not belong to the normal flow. Both data associations targeting flow nodes and sequence flows, as well as the data objects themselves are considered by the weaving algorithm, as they contain relevant information that contribute to the semantics of the process.
- Boundary events indicate that the activity to which they are attached should be interrupted when the event is triggered. The weaving algorithm reflects this by copying the aspect boundary events into the composed process.

Re-linking phase: After cloning the content of the advice (i.e., the flow nodes and connecting elements) in the context of the selected join point, the original connections of the join point potentially need to be re-routed to or from the newly cloned elements.

As a first step after the cloning is performed, the weaver needs to identify the elements that are candidates to be targets or sources of the so called re-linking phase. In analogy to the instantiation semantics of BPMN 2.0 we identify nodes with either only incoming or only outgoing edges as candidates. Yet, there may be the need for an explicit candidate specification, as shown in the unit testing advice of our example. Here, the contained gateway cannot be automatically detected as a final flow node of the advice process but this can be indicated using an end event. Note that these start and end events have to be removed at the end of the weaving process.

The re-linking strategy has to cover various composition setups, e.g., the combination of an advice with multiple tasks without successor and a selected join point with multiple outgoing sequence flows. An example of such a join point could be the *Create new Sprint backlog* task, as depicted in Figure 5. As an example aspect targeting this join point we consider a monitoring aspect that starts two timers after the backlog is initialized as shown in Figure 6.

In this situation (multiple outgoing edges and multiple end candidates), the re-linking strategy needs to merge the cloning structure with the original process and needs to keep the semantics, without overloading the composed model with edges. A full clone of the succeeding process fragment for each re-linking candidate of the advice would not be a practically scalable solution. To tackle this problem we leverage two BPMN 2.0 equivalences. In fact, the BPMN 2.0 specification defines an equivalence relationship between multiple outgoing sequence flows and a split using a parallel gateway as well as an equivalence relationship between multiple incoming sequence flows and a merge

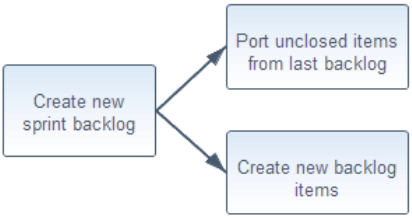

Fig. 5. Join point with multiple outgoing connections

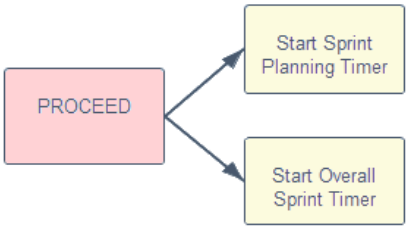

Fig. 6. Aspect with multiple end candidates

using an exclusive gateway. Taking this into account, the weaver applies the following generic re-linking strategy: All cloned elements that are candidates for re-linking will be connected to the originally steaming out elements through the use of an exclusive gateway connected to a parallel gateway. Figure 7 depicts the result of applying this strategy to the monitoring aspect shown in Figure 6.

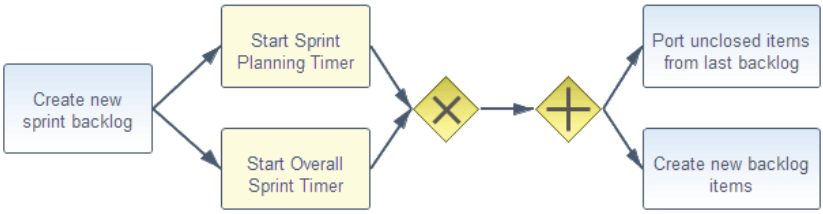

Fig. 7. Composed process in a scenario with multiple branches

The same strategy can be applied to multiple incoming join point edges and multiple re-linking start candidates. In the case of single edges and candidates we can avoid the creation of the intermediate gateways and simplify the re-linking using a direct connection via a sequence flow link.

3.2 Example Scenario Revisited

Figure 8 shows the result of weaving the TDD aspect shown in Figure 3 and the agile software development process depicted in Figure 2.

The integration of the first acceptance testing advice, the creation task, demonstrates the re-linking of multiple incoming sequence flows. As we do not have multiple start candidates in the advice there is no need to insert an additional merging gateway explicitly. The second acceptance testing advice with the *Run acceptance tests* task shows that not only activities but also other flow nodes can be valid join points. In this case the join point is a boundary timer event. The composition with a more complex process fragment is illustrated through the integration of the unit testing advice. Here, during the re-linking phase, the end event was replaced by the outgoing edges of the selected join point (i.e., the task *Implement*).

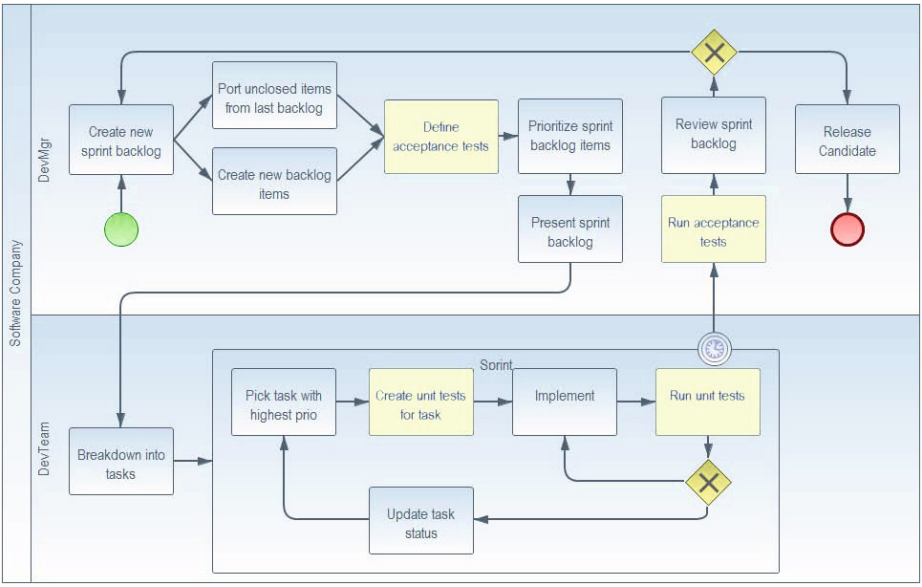

Fig. 8. Test-driven development process

4 Implementation and Tooling

Figure 9 gives a high level overview of the AO4BPMN toolset components. In the middle of this figure we see a *Repository*, which allows persisting and accessing aspect and process models. The *Navigator* can be used to browse processes and aspects; both types of models can be edited using a completely re-implemented new editor that extends the *BPMN2 Editor*. In addition, a *Weaving Wizard* is provided to support the selection of the base processes and aspects that should be composed by the weaver.

4.1 Editor

The editor shown in Figure 3 allows defining aspect models that are separated from the business process models, giving support to the AO4BPMN language. This editor is based on the tool BPMN2 modeler[1], which is an open source component of the Eclipse subproject Model Development Tools (MDT). The BPMN2 subproject aims at providing a metamodel implementation of the BPMN 2.0 specification and a corresponding modeler component. We extended and adapted the BPMN2 modeler to support AO4BPMN constructs making use of the powerful extension point provided by that tool. This extension point offers amongst others the following extension possibilities:

– Extending the property tabs of any modeling element defined in the BPMN2 metamodel (the actual eclipse implementation). We used this to add an AO4BPMN

[1] `http://eclipse.org/bpmn2-modeler/`

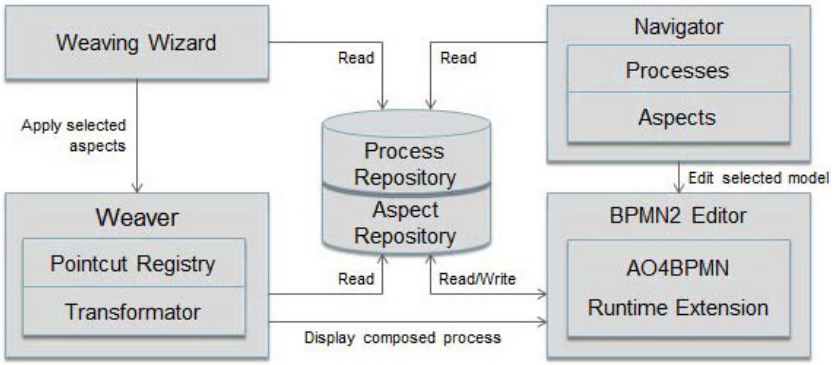

Fig. 9. Components of the AO4BPMN tooling

properties view tab that is displayed when an AO4BPMN-specific element is selected in the editor.
- Defining own custom tasks. We used this to define the *Proceed* element as a custom task.
- Defining own model extensions. That is, to extend the metamodel with new specialized BPMN elements. We used this extension point parameter to e.g., add the *Pointcut* attribute to the properties view of an *Advice*.
- Providing own feature containers for available modeling elements to override part of the default behavior. This offers a means to adapt the rendering and containment restrictions of e.g., *Advices*.
- Defining own style sheets. We used this to define styles for the adapted tool, e.g., the background color of the *Proceed* element.

The editor allows defining aspects, OCL-based pointcuts, and advice including also the proceed activity. A pointcut can be specified using the AO4BPMN property tab of an advice. Moreover, it is possible to extend the tool to add support for other pointcut languages. A mechanism to switch between multiple pointcut languages is already implemented.

4.2 Weaver and Weaving Wizard

As part of the toolset a weaver that implements the algorithms presented in Section 3 is integrated with the editor described above. Hence, the weaving process can be started directly from the editor. Furthermore, a weaving wizard is provided to support the selection of the processes and aspects that should be composed by the weaver. As shown in Figure 10, the weaving wizard allows selecting on the left side the diagrams containing the aspects to be woven and on the right side the diagrams containing the base processes. If the aspects are in the same diagram as the base processes the checkbox with the label *Use the same Diagrams* should be ticked. Once the button labeled *OK* is clicked the wizard calls the weaver.

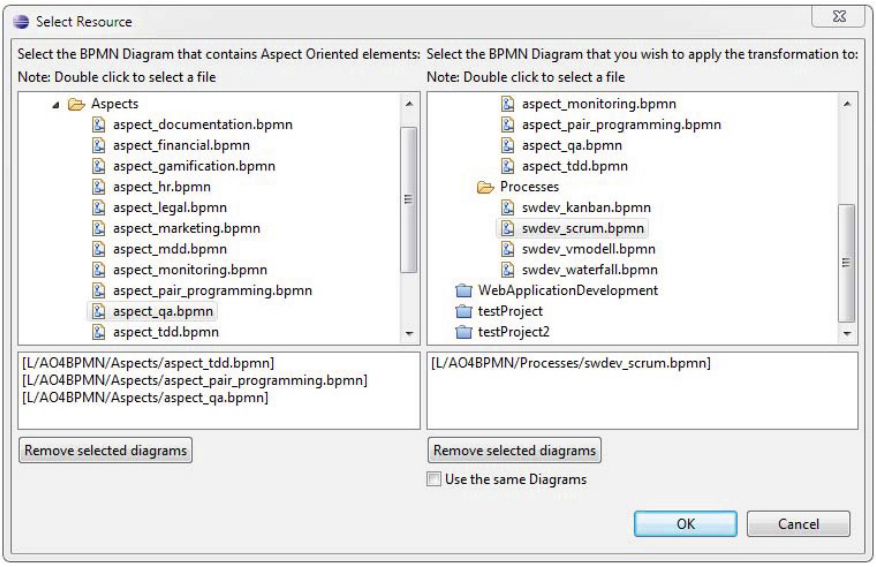

Fig. 10. Weaving wizard (Selection of base processes and aspects)

For implementing the weaver we used operational QVT[2], which is part of the OMG standard QVT. The reason for choosing QVTo and no other parts of the QVT language family is the operational nature of QVTo, which helped us in implementing the clone and merge algorithm in a rather straight-forward manner.

5 Related Work

We discuss in this section related work from two areas. The first area addresses the general topic of aspect-oriented modeling mainly in the context of object-oriented design. The second area addresses specifically the topic of aspect-oriented business process management.

An extensive survey on aspect-oriented modeling approaches is presented in [2], which is a deliverable of the AOSD-Europe network of excellence. That survey covers both works that extend UML diagrams with aspect-oriented constructs and works that are based on domain-specific languages. Another more recent survey is presented in [14] and it focuses specifically on approaches that are based on UML. As examples of works in that context we mention [16] and [6]. In [16], the authors present an aspect-oriented extension to UML state machines. In [6] an aspect-oriented extension to UML activity diagrams is proposed. Another similar work in this area is [10], which proposes a UML 2 profile for aspect-oriented modeling and uses UML 2 actions to define advice behavior in a platform independent manner. In addition that work defines an

[2] `http://projects.eclipse.org/projects/modeling.mmt.qvt-oml`

advanced pointcut language and a model weaving mechanism based on model transformation. The works in this first area do not address business process languages such as WS-BPEL and BPMN.

One of the first works in the second area is AO4BPEL [4], which is an aspect-oriented extension of the executable business process language BPEL. In that work, we proposed an aspect-oriented extension to BPEL 1.1 and developed an aspect-aware orchestration engine, which supports dynamic weaving. AO4BPEL join point model includes all BPEL activities and it uses XPath as pointcut language. A new implementation of that language based on Apache ODE was recently made available [9]. Padus [1] is another aspect-oriented extension to BPEL, which has a similar join point model and advice language. However, Padus uses a high-level logic-based pointcut language to support stateful aspects and express temporal pointcut expressions. Furthermore, Padus performs static weaving using model transformation unlike AO4BPEL, which supports dynamic weaving using an extended orchestration engine. AO4BPMN introduces aspect-oriented concepts to graphical business process modeling languages whereas AO4BPEL and Padus target the process execution language BPEL.

The authors of [7, 8] build upon AO4BPMN 1.0 [5] by proposing a formal definition of the different language concepts. They also propose a formal specification of the execution semantics of a weaving mechanism using colored petri nets. These works nicely complement AO4BPMN by providing a formal foundation. Furthermore, the authors of [8] also validate the usefulness of AO4BPMN in practice by presenting a banking case study. Another validation of AO4BPMN in the context of business software can be found in [15], where we use aspects behind the scenes to modularize and manage business process extensions as first-class entities in an industrial context. In that work, aspects are used as means to support extensibility and encapsulate business process extensions which are developed by software companies (e.g., as add-ons) that extend standard business applications. Unlike in that work, in the current paper we advocate an explicit approach to aspect-oriented business process modeling.

6 Conclusion

To improve the modularization of crosscutting concerns and process extensions in business process models, we proposed in a previous work an aspect-oriented extension to BPMN called AO4BPMN. However, that extension had some open issues especially with respect to the lack of a concrete pointcut language and the lack of a weaving mechanism. Furthermore, it was based on an older version of BPMN as BPMN 2.0 was still under development at that time. In the current paper, we addressed these issues and presented a concrete OCL-based pointcut language for AO4BPMN as well as a powerful weaver based on model-to-model transformation. We also refined the language definition to make it compatible with BPMN 2.0. In addition, we presented a new Eclipse based tooling for AO4BPMN including a graphical editor and a weaver. An application of an AO4BPMN variant for modularizing process extensions was presented in [15]. Another application in the context of a banking case study was presented in [8]. Our future work will focus on mapping AO4BPMN aspects to AO4BPEL as an alternative to weaving by model transformation and also on facilitating the definition of pointcuts to users that are not familiar with OCL for instance by providing a form-based query builder.

Acknowledgments. The work presented in this paper was performed in the context of the Software-Cluster project SINNODIUMIt was partially funded by the German Federal Ministry of Education and Research under grant no. 01IC12S01. The authors assume responsibility for the content.

References

[1] Braem, M., Gheysels, D.: History-Based Aspect Weaving for WS-BPEL Using Padus. In: Proc. of the 5th European Conference on Web Services (ECOWS), pp. 159–167 (November 2007)
[2] Brichau, J., et al.: Report describing survey of aspect languages and models, AOSD-Europe Deliverable D12
[3] Charfi, A.: Aspect-Oriented Workflow Management. VDM Verlag Dr. Müller (2008)
[4] Charfi, A., Mezini, M.: AO4BPEL: An aspect-oriented extension to BPEL. World Wide Web Journal: Special Issue: Recent Advances in Web Services 10(3) (March 2007)
[5] Charfi, A., Müller, H., Mezini, M.: Aspect-oriented business process modeling with AO4BPMN. In: Kühne, T., Selic, B., Gervais, M.-P., Terrier, F. (eds.) ECMFA 2010. LNCS, vol. 6138, pp. 48–61. Springer, Heidelberg (2010)
[6] Cui, Z., Wang, L., Li, X., Xu, D.: Modeling and Integrating Aspects with UML Activity Diagrams. In: Proc. of the ACM Symposium on Applied Computing, SAC 2009, pp. 430–437. ACM, New York (2009)
[7] Jalali, A., Wohed, P., Ouyang, C.: Aspect oriented business process modelling with precedence. In: Mendling, J., Weidlich, M. (eds.) BPMN 2012. LNBIP, vol. 125, pp. 23–37. Springer, Heidelberg (2012)
[8] Jalali, A., Wohed, P., Ouyang, C., Johannesson, P.: Dynamic weaving in aspect oriented business process management. In: Meersman, R., Panetto, H., Dillon, T., Eder, J., Bellahsene, Z., Ritter, N., De Leenheer, P., Dou, D. (eds.) ODBASE 2013. LNCS, vol. 8185, pp. 2–20. Springer, Heidelberg (2013)
[9] Look, A.: Ao4bpel 2 server (2011), https://github.com/alook/ao4bpel2
[10] Mosconi, M., Charfi, A., Svacina, J., Wloka, J.: Applying and Evaluating AOM for Platform Independent Behavioral UML Models. In: Proc. of the 12th International Workshop on Aspect-Oriented Modeling, AOM 2008, pp. 19–24. ACM, New York (2008)
[11] OASIS: Web Services Business Process Execution Language Version 2.0., http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html
[12] OMG: Object constraint language (ocl) version 2.3 (2012), http://www.omg.org/spec/OCL/2.3.1/
[13] Object Management Group (OMG): Business Process Model and Notation (BPMN) Version 2.0., http://www.omg.org/spec/BPMN/2.0/
[14] Wimmer, M., Schauerhuber, A., Kappel, G., Retschitzegger, W., Schwinger, W., Kapsammer, E.: A Survey on UML-based Aspect-oriented Design Modeling. ACM Comput. Surv. 43(4), 28:1–28:33 (2011), http://doi.acm.org/10.1145/1978802.1978807
[15] Witteborg, H., Charfi, A., Aly, M., Holmes, T.: Business Process Extensions as First-Class Entities — A Model-Driven and Aspect-Oriented Approach. In: Liu, C., Ludwig, H., Toumani, F., Yu, Q. (eds.) ICSOC 2012. LNCS, vol. 7636, pp. 763–770. Springer, Heidelberg (2012)
[16] Zhang, G., Hölzl, M., Knapp, A.: Enhancing UML State Machines with Aspects. In: Engels, G., Opdyke, B., Schmidt, D.C., Weil, F. (eds.) MODELS 2007. LNCS, vol. 4735, pp. 529–543. Springer, Heidelberg (2007)