Translating BPMN Models into UML Activities

María Agustina Cibrán

THALES
ThereSIS Security & Information Systems
Route Départamentale 128 - 91767 Palaiseau Cedex - France
maria-agustina.cibran@thalesgroup.com

Abstract. In business-driven development, IT architects must face the challenge of building IT solutions that are aligned with business processes in order to satisfy business requirements. To this endeavor, they must understand those business processes, which requires extensive collaboration with domain experts and analysts. Because business processes are typically expressed using dedicated modeling notations - e.g. BPMN - IT architects are forced to master their particularities. In order to support IT architects in performing the business-IT alignment, an approach for the automatic translation from BPMN models to UML activity models is proposed. By means of re-expressing business processes in a language that is closer to the architecture, the synchronization of the process models with other UML architectural models is enabled. In addition, architects are relieved from having to master BPMN. This paper reports on the challenges of defining and implementing such a translation. ATL is used as the model transformation language.

Keywords: BDD, BPMN, UML Activities, Model Transformations, ATL, MDA, MDE.

1 Introduction

Business Process Modeling [1] aims at representing the processes of a domain or business by creating dedicated business process models. It is an activity that is typically performed by business analysts and managers with the objective of improving clarity and accuracy of business requirements. It also contributes to realizing a more efficient communication between domain and IT experts. In addition, having explicit business process models enables the improvement of the processes efficiency and quality. A business process model typically represents interrelated business organizations and roles as well as it depicts the flow or progression of activities - tasks and subprocesses - each of which representing the work of a person, an internal system, or the process of a partner company towards some business goal. Nowadays, one of the most popular standard languages for the representation of business process models is the Business Process Modeling Notation (BPMN) [2].

In Business-Driven Development (BDD) [3] the goal is to develop IT solutions that directly satisfy business requirements and needs. Because the starting step

D. Ardagna et al. (Eds.): BPM 2008 Workshops, LNBIP 17, pp. 236–247, 2009.

for BDD is the creation of business process models, aligning business processes and IT solutions is crucial. This alignment implies the need for adapting the methodology used to derive the IT solution, to be able to use business process models as input artifacts to the design and development phases of the software development life cycle. However, aligning business processes to existing software methodologies is not an easy task: it does not only imply the need for understanding the business goals and expectations but it also requires taking decisions about which parts of the business processes are automated - vs. manually performed tasks - and clearly defining and realizing non-functional properties. These steps typically imply the need for extensive collaboration and communication efforts between domain experts, business and functional analysts, and IT architects. Moreover, in the effort of understanding business processes, IT architects are confronted with the particularities - i.e. notation and semantics - of the adopted business process modeling notation. Also, IT architects are forced to reason in terms of *process-centric* abstractions, which do not necessarily fit into the architectural abstractions, typically centered around concepts such as *object*, *component* and *service*.

In this article, an approach for BDD is proposed which relies on principles of Model-Driven Engineering (MDE). This approach aims at facilitating the alignment between business processes and IT solutions by means of automating the translation from BPMN models to UML activity models. Because UML has become the de-facto industry standard for modeling software systems, architectural models of contemporary industrial solutions typically rely on its formalisms and modeling support. We claim that re-expressing - in an automated way - the BPMN business process models in the form of UML activity models can (i) help improve IT architects' understanding of those business processes, (ii) enable the synchronization of those business processes with other architectural models expressed in UML, (iii) relieve IT architects from having to master BPMN's notation and formalisms. In addition and contrary to UML, no standard BPMN metamodel exists today which complicates BPMN's practical adoption in the context of MDE for BDD. The re-expression of business processes in UML can enable the subsequent definition of model transformations that rely on the standard UML metamodel.

The main contributions of this approach are: (i) the definition of a mapping from BPMN concepts to corresponding UML activity elements, and (ii) the implementation of this mapping in the form of a model-to-model transformation using ATL [8]. Defining a conceptual mapping between these two languages is not trivial - due to their different purposes and semantics. As extra contributions, non-straightforward mapping scenarios are identified (iii) and ATL's features are evaluated against the realization of those scenarios (iv).

This paper is organized as follows: Section 2 introduces BPMN and UML activity models, Section 3 presents the proposed conceptual mapping between the two, describing non-straightforward mapping scenarios for which ATL solutions are proposed in Section 4. Sections 5 and 6 present related and future work respectively, and conclusions are summarized in Section 7.

2 Selected Approaches to Business Process Modeling

The Business Process Modeling Notation (BPMN) is a standard graphical notation - initially developed by the Business Process Management Initiative (BPMI) and currently being maintained by the Object Management Group (OMG) - for the representation of business process models. BPMN was developed with the aim of having a notation that is understandable by all business stakeholders who typically manage and monitor those processes and to facilitate the link towards a technical implementation of the processes. BPMN graphical models typically depict the control flow of activities, the organizations and roles involved and the message exchange between them. Among the most common BPMN concepts we can mention *activities* (e.g. *tasks* and *subprocesses*) which are the units of behavior, *swimlanes* (e.g. *pools* and *lanes*) that help organize activities into roles and interacting parties involved in the process, *events* that occur during the process and *gateways* for expressing sequence flow. For more details on BPMN, the interested reader is referred to [2]. The choice of BPMN as business process language is driven by current needs from THALES customers manifesting their interest for modeling business interactions explicitly.

UML provides support for process modeling in the form of *Activity* models. Although this support is commonly known as UML *Activity Diagrams (ADs)*, in this article we adopt the name of UML *activities* or *activity models* to put the focus on the modeling capabilities and not on the graphical details. Among the concepts supported by UML, we can mention the concept of *activity* used to depict the overall behavior of a system, *actions* which encapsulate business behavior and are used to either represent business activities or business events, and *control nodes* used to depict typical control flow patterns. The suitability of UML activities for business process modeling has been proven by existing approaches [14,15,12]. Existing work also identifies limitations of this modeling language when it comes to representing resource-related or organizational aspects [14]. More information on UML activities can be found in [6] (Section 12, p.297).

It is important to note that while BPMN focuses on business processes, UML focuses on software design and therefore the two are not competing notations but allow to represent different views on systems [5].

3 A Conceptual Mapping from BPMN to UML Activities

This section reports on the proposed conceptual mapping from BPMN models to UML activity models. This mapping considers BPMN elements at the source side - as described in the adopted BPMN specification (v1.0) [2] - and UML elements at the target side, in particular the portion of the UML metamodel specification (v.2.1.2) corresponding to *activities* [6] (Section 12, p.297). The reason why the BPMN specification is used is the current lack of a standard BPMN metamodel[1]. Note that only a portion of the BPMN specification

[1] The Business Process Definition Metamodel (BPDM) aims at addressing this issue. More information on BPDM can be found at http://modeldriven.org/web/bpdm.

is considered in this mapping, namely common BPMN elements: *business process diagram*, *swimlanes*, *activities*, *sequence* and *message flows*, *gateways* and *events*. However, more advanced BPMN features such as transactions, multiple instances, exception handling and data flow have not been addressed yet. Mapping solutions for these features would require the investigation of more advanced solutions that rely on the identification and automatic detection of sophisticated patterns. The proposed mapping aims at preserving the same behavior, granularity and level of abstraction expressed in the source model. Thus, the target activity model aims at being an equivalent representation of the original BPMN model. Note that due to limited space, the complete mapping is not shown here. Instead, the rest of this section presents representative mapping scenarios which are of interest from the point of view of model transformation.

3.1 One-to-One Mappings

A natural approach for the definition of this mapping is to consider, for each of the BPMN elements, a single UML correspondent which complies with the semantics of that BPMN element. However, a one-to-one mapping can only be established for those elements for which a direct counterpart - a *UML Activity* metaclass - exists. Figure 1 gives examples of one-to-one mappings. For example, *pools* in BPMN are used to partition the process. A *pool* represents a participant involved in the process, such as a business entity. The *UML Activity* metaclass named *ActivityPartition* allows to group actions that have some characteristics in common. An equivalence can be drawn between these two concepts. The same analysis can be done for the other examples in the figure.

BPMN element	UML Activity metaclass
Pool	ActivityPartition
Lane	ActivityPartition
Activity	ActivityNode
Task	OpaqueAction
Subprocess	StructuredActivityNode
SequenceFlow	ControlFlow

Fig. 1. Examples of mappings denoting a single UML counterpart for each individual BPMN element

While defining these mappings we observe that for some BPMN elements it is not possible to find a unique distinct *UML Activity* metaclass that closely denotes its semantics. This is the case for e.g. the BPMN *Pool* and *Lane* elements, which represent participants and roles respectively. In the UML Activity metamodel, only the concept of partition is available, not distinguishing between the two different ways of partitioning a model. Thus, as a workaround, we are forced to map BPMN pools and lanes to the same UML concept of activity partition - embodied by the *ActivityPartition* metaclass.

The same situation is observed in the case of BPMN events. One of the main differences between BPMN and more traditional modeling notations is that BPMN provides extensive means to describe event-driven behavior. We observe however that some - very specific - types of events cannot have a straightforward mapping in terms of *UML Activity* concepts, because UML's support for events is much more limited than that of BPMN. Thus for example, the event types *error*, *cancel*, *compensation*, *link*, *terminate* and *multiple*, each of them with distinct semantics, are represented at the UML side by the same metaclass *FlowFinalNode*.

The reason for this mapping mismatch lies on the different focus and level of expressivity of the mapped languages. As a consequence, elements with a more specific semantics get mapped to elements with more generic semantics. Note that these mappings do not ensure the preservation of the full BPMN semantics. To overcome this limitation, an approach based on the semantical enrichment of UML - by means of, for instance, the definition of a UML profile - could be explored. This is subject of future work.

3.2 Mapping *Rich* BPMN Elements

Some BPMN elements have *rich* semantics because they embody different meanings in the same single concept. This is the case of, for example, BPMN events. For instance, a BPMN *start message event* combines the semantics of *starting the process* with *accepting a message*. The UML Activity metamodel however does not offer a metaclass that combines these two meanings. The UML metaclass *InitialNode* denotes a control node at which the flow starts when the activity is invoked; in turn, the metaclass *AcceptCallAction* denotes an accept event action representing the receipt of a synchronous call request. Thus, to overcome this mismatch we propose a solution that involves combining many *UML Activity* elements for the representation of a single BPMN element. In our example, we can define the counterpart of a *start message event* as a sequence composed of an *InitialNode* followed by an *AcceptCallAction*.

3.3 Mapping *Overloaded* BPMN Elements

We observe that certain BPMN elements are *overloaded* since they can have different meanings according to the way they are used in a model. This means that their semantics cannot be analyzed by looking at each of those elements in isolation but also at their relations with other model elements.

This is the case for example of BPMN *gateways*. In BPMN, the context in which the gateway is used determines different interpretations of the gateway and thus a different mapping to UML *control nodes*. According to the BPMN specification, a gateway can have multiple incoming and outgoing sequence flows which determine a different role of that gateway in the process. For example, an *OR gateway* can play the role of a *fork node* when it is used to split the control flow (i.e. multiple outgoing sequence flows) or it can play the role of a *merge node* when it is used to join different control flows (i.e. multiple incoming

BPMN element	Context	Semantics	UML metaclass
OR Gateway	incoming = 1 & outgoing > 1	*SPLIT*	ForkNode
OR Gateway	incoming > 1 & outgoing = 1	*JOIN*	MergeNode
OR Gateway	incoming > 1 & outgoing > 1	*SPLIT + JOIN*	ForkNode
AND Gateway	incoming = 1 & outgoing > 1	*SPLIT*	ForkNode
AND Gateway	incoming > 1 & outgoing = 1	*JOIN*	JoinNode
AND Gateway	incoming > 1 & outgoing > 1	*SPLIT + JOIN*	ForkNode
XOR Gateway	incoming = 1 & outgoing > 1	*SPLIT*	DecisionNode
XOR Gateway	incoming > 1 & outgoing = 1	*JOIN*	MergeNode
XOR Gateway	incoming > 1 & outgoing > 1	*SPLIT + JOIN*	DecisionNode

Fig. 2. Contextual interpretation of BPMN gateways and their mapping to UML control nodes

sequence flows). A similar analysis can be done for the *AND* and the *XOR gateways*. Figure 2 summarizes the different semantical interpretations of BPMN gateways and the UML correspondent for each contextual case. As a conclusion, the mapping of overloaded BPMN elements differs according to contextual information.

4 Automating the Mapping Using ATL

THALES is involved in the MODELPLEX project[2] the aim of which is to demonstrate the value of using MDE technologies for tackling the increasing complexity of software systems. One of the goals of THALES as industrial partner in MODELPLEX is to assess the maturity and suitability of MDE tools - including model transformation languages - for the engineering of industrial solutions for complex systems. In this context, THALES implemented a model transformation of the conceptual mapping proposed in Section 3. For this implementation, the BPMN metamodel which is implemented as part of the Eclipse SOA Tools Platform project [7] was chosen.

The ATLAS Transformation Language (ATL) [8] is a model transformation solution - including a language and development environment - for the implementation of model transformations. ATL is developed by the ATLAS Group (INRIA & LINA) and it is now one of the major components of the M2M Eclipse project. ATL is hybrid because it allows mixing two programming styles, declarative and imperative. ATL rules define how source model elements are matched and navigated to create and initialize the elements of the target models. More information on ATL can be found in [8].

In the rest of this section, we report on the experiences of using ATL for the implementation of the translation from BPMN to *UML Activities*. Solutions are

[2] More information on MODELPLEX can be found at http://www.modelplex.org

proposed for the mapping cases described in Section 3. Sections 4.1, 4.2 and 4.3 revisit the mapping scenarios described in Sections 3.1, 3.2 and 3.3, showing how ATL can be used to realize them, as well as pointing out challenges.

4.1 One-to-One Mappings

Implementing one-to-one mappings in ATL is done by defining matching rules that receive as input - i.e. *from* clause - a single BPMN element and produce as output - i.e. *to* clause - a single UML element. This implementation is straightforward.

With respect to the challenge posed by mapping different BPMN elements to the same UML element, as this is a conceptual mismatch, a solution cannot be found at the technological level. In order to avoid loose of information, and preserve the full semantics of BPMN elements, two solutions are envisaged (not explored yet): i) define complex UML patterns, ii) extend UML's semantics by using a profile approach that allows annotating the target model elements with extra semantical information. Exploring these alternatives are subject of future work.

4.2 Single Source to Composed Target

ATL matched rules allow specifying how target model elements must be generated from source model elements. To this end, a matched rule defines which source model element must be matched (*from* clause), the number and type of the generated target model elements (*to* clause) and the way these target model elements must be initialized from the matched source elements (in the rule's body).

Many different target model elements can be generated from a single source model element, feature that can be used to realize the translation of a single BPMN element to many UML elements. These target elements are linked together in a sequence by setting their attributes accordingly. An extract of this solution for the example of the *end message event* is shown below:

```
rule EndMessageEvent {
  from i: bpmn!Activity (i.activityType = #EventEndMessage)
  to ssa: UML!SendSignalAction (
      ...
      target <- inputpin ),
    inputpin: UML!InputPin (...),
    cf: UML!ControlFlow (
      ...
      source <- ssa,
      target <- ffn ),
    ffn: UML! FlowFinalNode(...)
}
```

This solution poses challenges at the implementation level: translating to several target elements does not only have a local impact - in the scope of the involved transformation rule - but it also impacts other transformation rules because they need to pick the right element among the many generate target

elements. To illustrate this impact, consider a BPMN sequence flow *sflow1* that has as target element an end message event *event1*. According to the mapping (Section 3), *sflow1* translates to a UML control flow *cflow1* whereas *event1* translates to a sequence composed of a send signal action *ssaction1*, a control flow *cflow2*, and a flow final node *ffnode1*. Thus, the transformation rule in charge of translating *sflow1* needs to assure that the generated *cflow1* has as target element the same send signal action *ssaction1*. This example shows that when single sources are translated to composed targets, transformation rules become dependent on each other, reducing this way the modularity of the implementation. Using ATL, these dependencies can be tackled by adding extra code in the dependent rules. This is however not ideal, as the realization of the dependency results in crosscutting code, violating separation of concerns. More advanced support is needed in transformation languages to better realize separation of concerns.

4.3 Single Source, Different Contextual Interpretations

In Section 3.3, the situation in which mapping a BPMN element depends on contextual information was described. ATL allows to implement these extra contextual conditions as part of the *from* clause of rule specifications. Thus for example, implementing the mapping of *OR gateways* can be done as shown below. Helpers are used in ATL to implement those conditions in modules that can be invoked from rules.

```
helper context bpmn!Activity def: usedAsSplit(): Boolean =
  self.incomingEdges -> size() = 1 and self.outgoingEdges -> size() > 1;
helper context bpmn!Activity def: usedAsJoin(): Boolean =
  self.incomingEdges -> size() > 1 and self.outgoingEdges -> size() = 1;
helper context bpmn!Activity def: usedAsSplitAndJoin(): Boolean =
  self.incomingEdges -> size() > 1 and self.outgoingEdges -> size() > 1;
rule ORGatewaySplit {
  from i: bpmn!Activity (i.activityType = #GatewayDataBasedInclusive
                         and i.usedAsSplit())
  to o: UML!ForkNode(...)}
rule ORGatewayJoin {
  from i: bpmn!Activity (i.activityType = #GatewayDataBasedInclusive
                         and i.usedAsJoin())
  to o: UML!MergeNode(...)}
rule ORGatewaySplitAndJoin {
  from i: bpmn!Activity (i.activityType = #GatewayDataBasedInclusive
                        and i.usedAsSplitAndJoin())
  to o: UML!ForkNode(...)}
```

Adding contextual conditions to transformation rules as a solution for distinguishing between different interpretations of source elements is suitable when only a few conditions need to be considered and when those conditions are based on information that is directly accessible from the source element. In the example shown above, all incoming and outgoing edges are accessible from the OR

gateway. However, when more global context that is not directly accessible from the source element needs to be inspected or when the number of conditions is big, then this solution falls short. More advanced support is needed in transformation languages to better deal with these issues.

In this section we reported on the experience of implementing the automatic translation from BPMN to UML Activities using ATL. A complete translation has been implemented and tested with real-world models from complex domains being developed in the THALES' MODELPLEX case study.

5 Related Work

State-of-the-art on BDD typically aims at translating business processes directly into executable implementations. Today, the Business Process Execution Language (BPEL) is the standard for representing executable business process behavior based on Web services [9], and thus it is a natural candidate for the implementation of workflow-based business processes. However, the translation of BPMN to BPEL is far from trivial due to conceptual mismatch. Some approaches exist today that analyze this mismatch and realize the automatic translation for certain scenarios [10].

However, while some approaches pursue a one-step translation to executable processes that can easily be triggered by business analysts, others consider this goal as 'one of the fallacies of the business process management' [11]. The reasons for this disbelief is that it is observed that most business processes typically require IT involvement in order to derive a suitable implementation. Moreover, having a direct translation from business processes to executable processes also assumes the existence of a one-to-one mapping between the abstractions of business processes on the one hand (e.g. tasks, subprocesses) and their implementing Web services of the BPEL process on the other hand, creating a tight conceptual coupling between the business and implementation spaces. The limitations of the one-step translation motivate the need for a more advanced approach that considers having an architectural layer in between the two, THALES' subject of research in MODELPLEX.

The mapping of BPMN and UML activities has been explored by other existing approaches. In [12] a conceptual mapping between these two languages based on the workflow patterns (as identified by [13]) is proposed. This work shows how these two graphical process modeling notations can be used to represent workflow patterns. The notations are evaluated with respect to technical ability and readability and several alternative - graphical - solutions for each pattern are proposed. Thus the focus is put on the expressivity of the languages for realizing the pattern, rather than on how the solutions can be mapped to each other (semi-)automatically. In [13,14,15], Van der Aalst et al. use the workflow patterns as an analysis framework. They provide a survey of the expressive power and features of process modeling languages that are suitable for the realization of the patterns - among them BPMN and UML - identifying similarities and differences between them. However, the actual automatic mapping between BPMN and UML is not investigated.

In [16], Murzek et al. point out that transformation languages are too generic to be suitable for solving more domain-specific transformation issues. They identify a list of problems when transforming business process models expressed in different languages, among them BPMN and UML activity diagrams and where the focus is put on basic control flow patterns. The authors also identify challenges in the implementation of automatic transformations between business process languages and use ATL as the transformation language.

In [17], Redding et al. explore a bottom-up approach for the transformation of object-oriented models to process-oriented models. The focus is put on control flow logic. The authors point out their interest on exploring the challenge of a top-down approach, as presented in this paper.

6 Future Work

Possible future directions of work include the investigation of solutions to overcome the challenge of translating - automatically or semi-automatically - entire workflow patterns, reusing conceptual solutions already proposed in state-of-the-art work [13,14]. Realizing this (semi-)automatic support is not trivial, as it involves first of all the automatic identification of patterns, which in turn requires solving the complex issue of delimiting the pertinent global context involved in the patterns. Also, from an implementation point of view, current transformation languages only allow the manipulation of individual model elements in the transformation rules. This is not enough to be able to modularize groups of elements involved in complex patterns.

Another direction of future work is the investigation of solutions to overcome more global semantical issues. For example, in both BPMN and UML activities, it is possible to define multiple start nodes for one process model. However, the semantics of initial nodes is different in both languages, as pointed out in [16]. In activity models, the start of the process implies the trigger of all initial nodes, whereas in BPMN, only the triggering of one start event can trigger the beginning of the process. Realizing a solution to this semantical issue would imply being able to capture and analyze global context. As illustrated earlier in the paper, the more global the context, the more dependencies there are between the transformation rules, which affects maintainability. More advanced transformation language features need to be investigated in order to facilitate this task. Moreover, in order to avoid loosing information at transformation time and preserve the full semantics of BPMN elements, more advanced solutions that rely on complex patterns and UML profiles for semantical enrichment are to be explored.

BPDM aims at defining a common metamodel for the various process notations, methodologies and standards, among them BPMN. Although this is a very promising initiative, it is still under development. Once available, the approach presented in this paper can be adapted to consider BPDM as the source metamodel.

Ongoing work is the definition of a MDE methodology for BDD. This methodology relies on the use of UML activity models as intermediate models in

between BPMN business process models and core architectural models of the IT solution - i.e. Platform Independent Model (PIM) and Platform Specific Model (PSM) in the context of Model-Driven Architecture (MDA) [4]. The goal is to use model transformations to support the entire software development process, moving from BPMN models towards architectural models and finally code artifacts. A special focus is put on the synchronization and traceability between models at the same and different levels of abstraction.

7 Conclusions

In this paper, an approach that facilitates the alignment between business processes and IT solutions was presented. To this end, a model transformation was proposed - including a conceptual mapping and implementation - to bridge from a business process model expressed in BPMN to an activity model expressed in UML. This paper also reported on the experiences in pursuing this automatic translation using ATL as the transformation language. The generated UML activity model can serve as a starting point for the definition of the core architectural steps. This work shows that the implementation of such a translation using contemporary model transformation languages such as ATL is possible to a certain extent. However, more challenging mapping scenarios were identified which motivate the need for more advanced model transformation support.

Re-expressing business processes in UML - the current de-facto modeling language typically familiar to most IT architects - can help improve IT architects' understanding of those business processes, relieving them from having to master BPMN. Moreover, implementing this translation in the form of a model-to-model transformation allows the encapsulation of expert and reusable transformation knowledge and relieves the need for manual work. In addition, in a typical architectural scenario where many interrelated UML models coexist, having business processes expressed in the form of UML activities can enable the synchronization and traceability between those different models.

Acknowledgments. This work has been carried out in the context of the MODELPLEX project contract number 034081 (co-funded by the European Commission under the IST Sixth Framework Programme). Thanks to Dhouha Ayed, Quentin Dallons, Didier Boulet, Ivo Bettens, Jérome Le Noir and Christophe Cubat for their interesting comments on the work presented in this article.

References

1. Havey M.: What Is Business Process Modeling (2005), `http://www.onjava.com/pub/a/onjava/2005/07/20/businessprocessmodeling.html`
2. OMG, Business Process Modeling Notation Specification v1.0. OMG Final Adopted Specification (February 2006), `http://www.bpmn.org/`
3. Mitra T.: Business-Driven Development. IBM DeveloperWorks (2005), `http://www-128.ibm.com/developerworks/webservices/library/ws-bdd/`

4. OMG, http://www.omg.org/mda/
5. BPMI.org, Business Process Modeling Notation (BPMN) Information, FAQs (2004), http://www.bpmn.org/Documents/FAQ.htm
6. OMG, Unified Modeling Language (UML), Version 2.1.2
7. Eclipse, SOA Tools Platform Project, BPMN Modeller, http://www.eclipse.org/stp/bpmn/
8. ATLAS Group (INRIA & LINA), ATLAS Transformation Language, http://www.eclipse.org/m2m/atl/
9. OASIS, Web Services Business Process Execution Language Version 2.0 (2007)
10. Ouyang, C., van der Aalst, W., Dumas, M., ter Hofstede, A., Mendling, J.: From Business Process Models to Process-Oriented Software Systems. In: ACM Transactions on Software Engineering Methodology (to appear); Preliminary version, http://eprints.qut.edu.au/archive/00005266/01/5266.pdf
11. Dubray J.: The Seven Fallacies of Business Process Execution. Article at InfoQ (2007), http://www.infoq.com/articles/seven-fallacies-of-bpm
12. White S.: Process Modeling Notations and Workflow Patterns. IBM (2004), http://www.bpmn.org/
13. Wohed, P., van der Aalst, W., Dumas, M., ter Hofstede, A., Russell, N.: On the Suitability of BPMN for Business Process Modelling. In: Dustdar, S., Fiadeiro, J.L., Sheth, A.P. (eds.) BPM 2006. LNCS, vol. 4102, pp. 161–176. Springer, Heidelberg (2006)
14. Russell, N., van der Aalst, W., ter Hofstede, A., Wohed, P.: On the Suitability of UML 2.0 Activity Diagrams for Business Process Modeling. In: Proceedings of the 3rd Asia-Pacific conference on Conceptual Modeling. Australian Computer Society, Inc. (2006)
15. Wohed, P., van der Aalst, W., Dumas, M., ter Hofstede, A., Russell, N.: Pattern-based Analysis of UML Activity Diagrams. In: Proceedings of the 25th International Conference on Conceptual Modeling. Springer, Heidelberg (2005)
16. Murzek, M., Kramler, G.: Business Process Model Transformation Issues - The Top 7 Adversaries Encountered at Defining Model Transformations. In: ICEIS, vol. 3, pp. 144–151 (2007)
17. Redding, G., Dumas, M., ter Hofstede, A., Iordachescu, A.: Transforming Object-Oriented Models to Process-Oriented Models. In: ter Hofstede, A.H.M., Benatallah, B., Paik, H.-Y. (eds.) BPM Workshops 2007. LNCS, vol. 4928, pp. 132–143. Springer, Heidelberg (2008)