Optional Activities in Process Flows

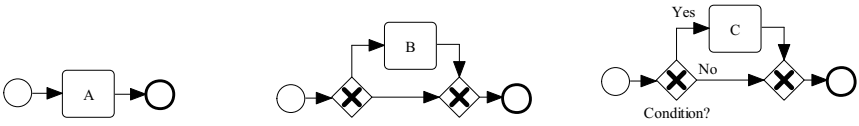
Christine Natschläger, Verena Geist, Felix Kossak, Bernhard Freudenthaler

Software Competence Center Hagenberg GmbH
Softwarepark 21
A-4232 Hagenberg
{firstname.secondname}@scch.at

Abstract: A drawback of many business process modelling languages (BPMLs) is that modalities are implicitly expressed through the structure of the process flow. All activities are implicitly mandatory and whenever something should be optional, the process flow is split to offer the possibility to execute the activity or to do nothing. This implies that the decision whether to execute one or more activities is described within another element, such as a gateway. The separation of decision and execution requires additional modelling elements and a comprehensive understanding of the entire process to identify mandatory and optional activities. In this paper, we address the problem and present an approach to highlight optionality in BPMLs based on the Control-Flow Patterns. Furthermore, we study the semantics of explicitly optional activities and show how to apply the general approach to a concrete BPML like the *Business Process Model and Notation* (BPMN). An explicitly optional activity is deemed to be more intuitive and also positively affects the structural complexity and the understandability of the process flow as shown by a graph transformation approach and a case study.

1 Introduction

Many business process modelling languages (BPMLs) provide a solely implicit expression of modality through the structure of the process flow. All activities are implicitly mandatory and whenever something should be optional, the process flow is split to offer the possibility to execute the activity or to do nothing. This implies that the decision whether to execute an activity is described within another element, e.g. a gateway. The separation of decision and execution requires additional modelling elements to split and merge the process flow and a comprehensive understanding of the entire process to identify mandatory and optional activities. For example, three process flows are shown in Fig. 1. In the first case, activity A must be executed in every process instance and is, thus, mandatory (see Fig. 1(a)). In the second case, the process flow is split to express that activity B is optional (see Fig. 1(b)). In addition, the modality also depends on the type of the split (user vs. conditional choice, parallel/exclusive/inclusive split) and on alternative elements. For example, the process flow shown in Fig. 1(c) has the same structure but comprises a conditional choice. In this case, activity C is mandatory if the condition evaluates to true (conditional obligation).

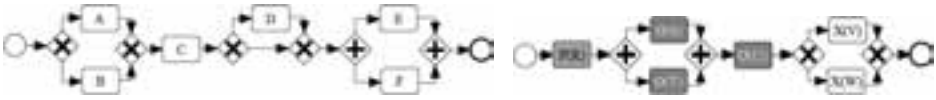


(a) Mandatory Activity (b) Optional Activity (c) Conditional Obligation

Figure 1: Implicit Expression of Modality through Process Flow Structure

This paper is also motivated by previous work (see [Nat11], [Nat12]), in which deontic logic was used to highlight modalities and applied to the *Business Process Model and Notation* (BPMN). The understandability of the approach was studied within a preliminary survey, which was answered by 22 post-graduate computer scientists. Although the number of respondents is too small for a significant survey, the preliminary survey provides some interesting results and identifies possible problems.

One representative example of the survey is shown in Fig. 2 and the explicit expression of modality is based on the deontic concepts of permission P (optional), obligation O (mandatory), and alternative X. The respondents then answered six questions for each model type as, for example, which tasks are mandatory, which tasks are optional, or what happens if task D (or R) cannot be executed. Note that in order to avoid a recognition of models expressing the same example, corresponding tasks received different names and the order of examples, elements, and questions varies.



(a) Implicit Expression of Modality (b) Explicit Expression of Modality

Figure 2: Preliminary Survey: Example

In summary, 64 mistakes emerged in the implicit and 15 mistakes in the explicit expression of modality, so the number of mistakes could be reduced by 77%. Considering the entire survey, further mistakes emerged in the deontic extension, because other examples comprised complex deontic constructs like conditional commitments and multiple deontic classifications, but still the overall number of mistakes could be reduced by 14%. In addition, all respondents were asked whether they prefer the implicit or explicit expression of modality. In most examples, the explicit expression was preferred, especially for more complex process flows. So the preliminary survey allows us to assume that a highlighting of modalities and a reduction of structural complexity increases the understandability of a process flow.

The following section provides an overview of related work and presents the expression of modality in other BPMLs like *UML Activity Diagrams* (UML ADs), *Event-Driven Process Chains* (EPCs), or *Yet Another Workflow Language* (YAWL). Subsequently, Section 3 identifies optional activities in BPMLs based on the *Control-Flow Patterns*, provides a description of the semantics based on *Abstract State Machines* (ASMs), and extends BPMN

with optional activities. The approach for optional activities is then evaluated in Section 4 by a graph transformation that shows a reduction of structural complexity and a case study taken from an industrial project. This papers concludes with a summary of the results and a description of future goals.

2 Related Work

There are several scientific investigations that highlight the importance and need for distinguishing between optional and mandatory behaviour in BPMLs. For example, in their reference model, Reinhartz-Berger et al. introduce stereotypes for constraining the multiplicity of elements within activity diagrams [RBSS05]. Optional activities can be denoted as $\ll$0..n$\gg$ or $\ll$0..1$\gg$, indicating that the activity can be performed any number of times or at most one time respectively. In [MA02], Muthig and Atkinson introduce a model-driven approach for product line engineering and define optional activities in a platform-independent way for capturing variabilities. They therefore apply the "VariantAssetElement" pattern and represent an optional activity by setting the element's fill colour to grey. Kïnzle and Reichert describe a data-driven approach for object-aware process management [KR11] that allows for realising optional activities, which enables authorised users to access object instances asynchronously to the normal control flow, i.e. to optionally execute activities at any point in time.

However, common languages and notations, such as BPMN, UML ADs, EPCs, or YAWL, do not provide means to explicitly denote optional and mandatory activities. For this reason, Kïnzle and Reichert suggest to add conditional branches at various positions within a process model as a workaround for supporting optional activities for process-oriented approaches [KR09]. Chang and Kim [CK07] also recommend this approach to realise optional activities for defining workflows in BPMLs in order to develop adaptable services. In the same way, Oestereich advises to express optional activities in UML ADs through the structure of the process flow [Oes02]. So all approaches suggest to express modality through the structure of the process flow. As a result, users have difficulties with comprehending the resulting bulky and spaghetti-like process models as well as distinguishing between optional and mandatory activities [KR09].

Besides some specialised versions of EPCs and YAWL, e.g. C-EPCs [Ros09] or C-YAWL [tHvdAAR10], that support optionality, there also exist several recommendations and extensions for handling optional activities in UML and BPMN. Razavian and Khosravi present a method for modelling variability in business process models based on UML ADs [RK08]. They propose modelling solutions for optional activities that are classified based on the origins of variability, i.e. control flow, data flow, and actions, and realised by multiplicities and stereotypes respectively for business process elements. In [PSWW05], Puhlmann et al. give an overview of variability mechanisms in product family engineering. They suggest to realise optional activities in UML ADs using variation points represented by "Null-Activities", i.e. a variant being an optional activity can be bound to a variation point using a dependency relation, both having assigned respective stereotypes.

Schnieders and Puhlmann [SP06] adapt the stereotype concept of UML to BPMN in order to address variability mechanisms for e-business processes families. They propose a ≪Null≫ variation point that can be resolved with specific variants to represent optional behaviour. In case that the variation point has only one optional resolution, the activity can be marked with an ≪Optional≫ stereotype and directly placed within the main process flow. A further approach for realising optional activities in BPMN suggested in [Mic10] is to apply ad-hoc sub-processes which leave the sequence and number of performances for a defined set of activities up to the performers (cf. [OMG11]). However, in that case more logic than just the concept of the completion condition is needed (see also [Sil09]). Using text annotations for displaying additional information about process elements, also suggested in [Mic10], has the drawback that annotations marking variabilities cannot be distinguished from other annotations in a business process diagram [PSWW05].

3 Optional Activities

In order to address the problems noted above, we propose to explicitly introduce optional activities in business process modelling. By optional activities we mean activities which a user can choose at runtime to either perform or not, without affecting control flow. Such activities should be graphically marked, either by a dashed line or by colour.

Activities are implicitly optional if they are, for example, defined after an exclusive choice or a multi-choice and if there is at least one alternative *Phi*-Path (empty path). With explicitly optional activities, the *Phi*-Paths and the respective choices can be removed, since the choice is directly expressed within the activity.

We first consider the Workflow Patterns introduced by research groups around Wil van der Aalst and Arthur ter Hofstede (see e.g. [vdAtHKB03], [vdAtH11]), because they are independent of concrete BPMLs and, thus, a solution for Workflow Patterns can be easily adapted for most concrete BPMLs, including BPMN, UML ADs, YAWL, and EPCs.

Next we compare the semantics of diagrams with conventional activities only and diagrams containing optional activities. Then we apply our concept to BPMN.

3.1 Control-Flow Patterns

In this section the Control-Flow Patterns are studied and optional activities are identified. An overview of all Control-Flow Patterns that express modality is given in Tab. 1. The first column provides the pattern number and the second column the pattern name. Only the exclusive and multi-choice pattern may express optionality and are, thus, studied in more detail in the following.

Table 1: Control-Flow Patterns with Modalities based on [vdAtH11]

No.	Pattern Name
Basic Control-Flow Patterns	
1	Sequence
2	Parallel Split
4	Exclusive Choice
Advanced Branching and Synchronization Patterns	
6	Multi-Choice
State-based Patterns	
16	Deferred Choice
18	Milestone
Iteration Patterns	
10	Arbitrary Cycles
21	Structured Loop
Trigger Patterns	
23	Transient Trigger
24	Persistent Trigger

3.1.1 *Exclusive Choice* Pattern

The *Exclusive Choice* pattern describes the divergence of one branch into two or more branches, but the thread of control is only passed to exactly one outgoing branch as shown in Fig. 3. The selection of the outgoing branch is based on an evaluation of the conditions defined for the outgoing paths [vdAtH11].

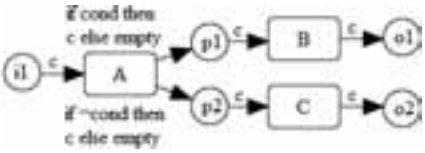


Figure 3: *Exclusive Choice* Pattern (Source: [vdAtH11])

The modality depends on the type of choice (user vs. conditional) and on a possible *Phi*-Path (i.e. an empty path that directly connects the split with the corresponding merge). If the exclusive choice provides a user decision with one or more *Phi*-Paths, then these *Phi*-Paths can be removed and all other tasks are classified as optional (marked with a dashed line) as shown in Fig. 4. If there is only one other path apart from the *Phi*-Path, then also the exclusive choice (split and merge) can be removed.

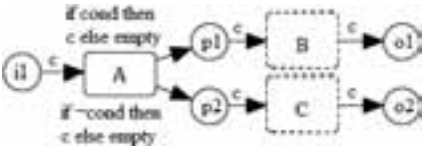
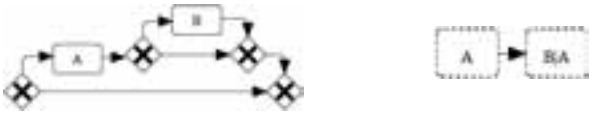


Figure 4: *Exclusive Choice* Pattern: Optional Activities

Conditional commitments (i.e. preconditions) are necessary for nested choices as shown by the BPMN example in Fig. 5(a) with the modality highlighted in Fig. 5(b). The precondition defines that task B may only be executed if task A was executed before.



(a) Example: BPMN (b) Example: Optional Activities

Figure 5: Nested Choice with Conditional Commitment

3.1.2 *Multi-Choice* Pattern

The *Multi-Choice* pattern describes the splitting of one branch into two or more branches, where the thread of control is passed to one or more branches as shown in Fig. 6. The selection of the outgoing branches is again based on an evaluation of the conditions defined for the outgoing paths [vdAtH11].

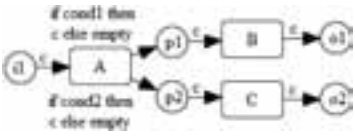


Figure 6: *Multi-Choice* Pattern (Source: [vdAtH11])

Similar to the exclusive choice pattern, the modality depends on the type of choice (user vs. conditional) and on a possible *Phi*-Path. If the multi-choice provides a user decision with one or more *Phi*-Paths, then these *Phi*-Paths can be removed and all other tasks are classified as optional (marked with a dashed line) as shown in Fig. 7. If there is only one other path apart from the *Phi*-Path, then also the multi-choice can be removed.

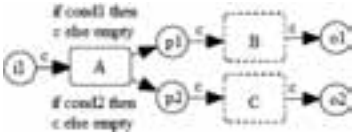


Figure 7: *Multi-Choice* Pattern: Optional Activities

3.2 Comparing the Semantics of Business Process Diagrams with and without Optional Activities

We now show that the introduction of optional activities, with the potential omission of certain choices (or gateways), is possible without altering the semantics of diagrams.

We show that a standard process diagram d on the one hand and a diagram d' with (explicitly) optional activities, which is derived from d by a respective transformation, on the other hand have the same semantics in the following sense:

1. Whenever a particular activity a would have been performed in d, also the corresponding optional activity a' in d' is performed and
2. A partial order with respect to the performance of activities in time is preserved, i.e., if it is relevant that activity a_1 is performed before a_2 in d, then also a'_1 is performed before a'_2 in d'.

We define the semantics with a token-based model. In conventional diagrams, an activity will be performed if it has been reached by a token and if certain other conditions are met (such as the availability of data or resources). We can express this algorithmically in the following way (using an *Abstract State Machine* (ASM) notation, see e.g. [BS03]):

```
WorkflowTransition(activity) =
  if controlFlowCondition(activity) and
        performanceCondition(activity) then
    ConsumeTokens(activity)
    PerformanceOperation(activity)
    wait until completed(activity) then
      ProduceTokens(activity)
```

where the `controlFlowCondition` signifies whether a token has reached the given activity and `performanceCondition` summarises all the other conditions for the activity to be able to perform (this simplification is sufficient for the given context of this paper). The three actions – `ConsumeTokens`, `PerformanceOperation`, and `ProduceTokens` – can be started in parallel (note that `ProduceTokens` is appropriately guarded; also note that we have omitted issues regarding different process instances for the sake of simplicity).

The semantics of an exclusive choice with two outgoing paths can be derived from `WorkflowTransition(activity)` by defining `ProduceTokens(activity)` as follows:

```
ProduceTokens(exclusiveChoice) =
  if choiceCondition(exclusiveChoice) then
    ProduceTokensAt(outgoingPlaces(exclusiveChoice)[0])
  else
    ProduceTokensAt(outgoingPlaces(exclusiveChoice)[1])
```

where `outgoingPlaces` is a list of places directly connected to the exclusive choice at the outgoing paths, out of which the first or, respectively, the second place is chosen. (We have renamed the argument to make it clear that this definition holds for exclusive choices only.)

The semantics of a respective exclusive merge (or "simple merge") can be derived by setting

```
controlFlowCondition(exclusiveMerge) =
  forsome (p in incomingPlaces(exclusiveMerge)) :
      enabled(p)
```

where `incomingPlaces` denotes the set of places relevant for the transition – i.e. (at least) one of the places before the choice must have the required number of tokens.

We now look at a subdiagram d with one incoming and one outgoing path, consisting of an exclusive choice c and an exclusive merge, joined by two paths, one of which contains one activity a, while the other path is empty (a *Phi*-Path).

The corresponding subdiagram d' derived from d by using (explicitly) optional activities will consist of a single optional activity a', with neither a choice (split) nor a merge. The semantics of such an optional activity can be modelled as follows:

```
WorkflowTransition(optionalActivity) =
  if controlFlowCondition(optionalActivity) and
        performanceCondition(optionalActivity) then
    ConsumeTokens(optionalActivity)
    if choiceCondition(optionalActivity) then
      PerformanceOperation(optionalActivity)
    if choiceCondition(optionalActivity) then
      wait until completed(optionalActivity) then
        ProduceTokens(optionalActivity)
    else
      ProduceTokens(optionalActivity)
```

Note that the definition of the semantics of a conventional activity as given further above can be derived from this definition for an optional activity by simply setting `choiceCondition(optionalActivity) := true`.

We assert (by design) that in d', `choiceCondition(a')` is identical to `choiceCondition(c)` in d (where c is the exclusive choice). Then it is easy to see that whenever a is performed in d, also a' will be performed in d'.

Also the control flow after the subdiagram remains unchanged, because if a' actually performs (i.e. `choiceCondition(a') = true`), then it sends a token (or tokens) on, just like a (in the latter case via the exclusive merge), and if a' does not perform, then it still sends on a token (or tokens), just like the *Phi*-Path in d would have done (again via the exclusive merge). Furthermore, if we assume that time for the splitting and merging activities of the exclusive choice and the exclusive merge can be neglected (in contrast to the performance of the main activity), then also the temporal performance of the subdiagram is unchanged. Therefore, also the sequence of activities in the wider context (i.e. including before and after the subdiagram) remains unchanged.

Within the given space, we can show the equivalence of the semantics only for such a simple case as that of d and d' discussed above. However, we think that the reader can already guess that this will also hold for more complex cases (as we have already checked for all relevant cases in yet unpublished work).

3.3 Optional Activities in BPMN

A BPMN activity can be either a *Task*, a *Sub-Process*, or a *Call Activity*. Although there is no formal basis for optional activities in BPMN (see also Section 2), process modellers should have the possibility to make use of optional activities to enhance intelligibility of their business process diagrams. This can be realised by introducing an explicit graphical representation of an optional activity, which must conform to the BPMN specification [OMG11]. According to this specification, an extended BPMN diagram may comprise new markers or indicators as well as new shapes representing a kind of artefact. In addition, graphical elements may be coloured where the colouring may have a specified semantics, or the line style of a graphical element may be changed if this change does not conflict with any other element.

Considering optional activities, we suggest the three representations shown in Fig. 8. In Fig. 8(a) an activity is marked as optional by an additional marker that corresponds to the modal operator of possibility ($\Diamond$). The advantage of this representation is that markers are a well-known concept in BPMN and easy to understand; however, since activities may comprise several markers, a further marker can result in an overcrowded representation. Thus, another possibility is to define a different background colour for an optional activity, e.g. green (see Fig. 8(b)). The advantage of this representation is that it is supported by most modelling tools; however, it requires diagrams to be coloured. The last suggestion is shown in Fig. 8(c) and uses a dashed line to highlight an optional activity. This representation is easy to understand and can be represented by several modelling tools. Note that the dashed line style does not conflict with the dotted line style of an event sub-process.

(a) Marker (b) Colour (c) Line Style

Figure 8: Possible Representations of Optional Activities in BPMN

4 Evaluation

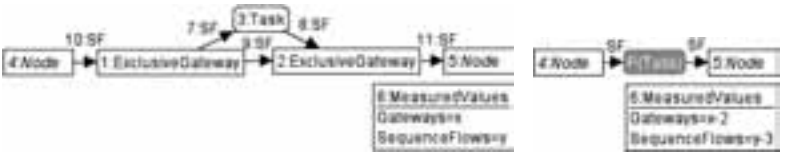
Finally, we evaluate our approach for optional activities by a graph transformation that shows a reduction of structural complexity and a case study taken from an industrial

project. The case study comprises a workflow implemented in an industrial project for the prevention domain of the *Austrian Social Insurance Company for Occupational Risks* (AUVA). Due to the fact that the original order execution process is too complex for an example in this contribution, we combined several details to sub-processes to better demonstrate the reduction of modelling elements and increased comprehensibility of diagrams. Furthermore, pools and lanes are not considered in this example. An extract of the order execution process and its transformation to an explicit representation of optionality is presented in [Nat11].

4.1 Graph Transformation

In order to study the reduction of structural complexity, we defined a graph transformation system (GTS) and specified several transformation rules to transform models with an implicit expression of modality to models with an explicit expression based on deontic logic. By adding only one binary attribute (optionality), the number of gateways and sequence flows can be reduced. Every transformation rule further highlights the original and the resulting number of gateways and sequence flows in an element called *MeasuredValues*. The GTS currently uses BPMN as a specific BPML to represent exclusive and multi-choices and is limited to structured diagrams with one task per path following a splitting gateway. In the following, only the transformation rules for exclusive choices are presented, since the multi-choice transformation rules are very similar.

ExclusiveWithPhiDualRule: The rule *ExclusiveWithPhiDualRule* takes an exclusive gateway with a task and a *Phi*-Path and transforms it to a permissible task (see Fig. 9). A negative application condition (NAC) forbids further alternative nodes. The transformation leads to a reduction of two gateways and three sequence flows.



(a) LHS (b) RHS

Figure 9: ExclusiveWithPhiDualRule

ExclusiveWithPhiRule: The next rule is called *ExclusiveWithPhiRule* and transforms a task of an exclusive gateway with a *Phi*-Path into a permissible task (see Fig. 10). The rule can be applied several times to classify an arbitrary number of tasks. A positive application condition (PAC) specifies that a further node exists, since otherwise the rule *ExclusiveWithPhiDualRule* should be applied. This transformation rule does not reduce the number of gateways and sequence flows.

(a) LHS (b) RHS

Figure 10: ExclusiveWithPhiRule

ExclusiveWithPhiRuleFinish: The rule *ExclusiveWithPhiRuleFinish* will be applied after the rule *ExclusiveWithPhiRule* and removes the *Phi*-Path as shown in Fig. 11. The left-hand side of the rule requires two permissible tasks as well as a *Phi*-Path, and a NAC forbids further not transformed tasks. The transformation leads to a reduction of one sequence flow.

(a) LHS (b) RHS

Figure 11: ExclusiveWithPhiRuleFinish

Since every transformation rule leads to equally many or fewer gateways and sequence flows, the structural complexity of the explicit representation is either the same or lower. The graph transformation system is described in more detail in [Nat11].

4.2 Case Study

In order to study the suitability of the proposed extension for the prevention domain of the AUVA, we provide a case study with a workflow taken from a business environment. The goal of the prevention domain is to suggest possibilities to improve the employees' safety and health conditions at their workplaces. For this purpose, the prevention workers receive requests to visit companies. Thus, one major workflow describes the process of an order execution. Since this workflow is typical for many companies, it is taken as a basis for this case study. The BPMN diagram of the order execution process is shown in Fig. 12. The process comprises the following activities: *Create Request* (CR), *Approve Order* (AO), *Appointment Management* (AM), *Order in Progress* (OP), *Execute Order* (EO), *Report Management* (RM), and *Close Order* (CO). The sub-processes *AM* and *RM* comprise further details.

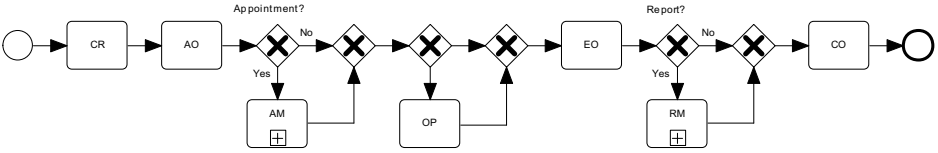


Figure 12: Order Execution Process: Implicit Expression

In a first step, the order execution process defines that a request is created (different roles are disregarded in this example). If a request was created, then this request must be approved resulting in an order. Afterwards, an appointment can be defined and it can be specified that the order is in progress. Subsequently, every order must be executed and reports can be created and modified. Finally, the order is closed. Although it is possible to discuss some aspects of the process flow, e.g., the optional creation of appointments or reports, this model describes an actual workflow taken from an industrial project based on real customer requirements (in a simplified form). For example, appointments are optional, since prevention workers may visit companies spontaneously if they are nearby. Considering the order execution process, all splitting gateways provide user choices and do not depend on any states or data. The entire diagram in this simple example consists of 6 gateways and 17 sequence flows. In the original form it is more complex (26 gateways and 64 sequence flows). Thus, it is even more difficult to identify optional and mandatory tasks. The explicit expression of the order execution process is shown in Fig. 13. Every complete order execution requires that a request is created and that the order is approved, executed, and closed, which is clearly shown in the proposed extension using general (mandatory) activities. All other activities are highlighted as optional.

Figure 13: Order Execution Process: Explicit Expression

The explicit expression provides two advantages with respect to understandability. First, the diagram only consists of 8 sequence flows and no gateways are necessary. So it was possible to remove 6 gateways and 9 sequence flows and thereby reduce the structural complexity of the process flow. Secondly, mandatory and optional activities can be distinguished at first sight based on the highlighting (e.g., different line style). It is still necessary to decide whether an optional activity is executed or not but instead of describing this decision through separate gateways and alternative paths, the decision is described within the corresponding activity. The claim that the understandability of the order execution process was increased is described in detail in a preliminary survey (see [Nat11], [Nat12]). In summary, this case study comprises an order execution process and demonstrates the transformation from an implicit to an explicit expression of modality. The structural complexity was reduced and the understandability increased.

5 Conclusion

In this paper, we addressed a drawback of many BPMLs, i.e. to implicitly express modalities through the structure of the process flow. This leads to additional modelling elements such as gateways or sequence flows, and thus complicates the identification of mandatory and optional activities. A preliminary survey affirmed that explicitly optional activities appear to be more intuitive and also positively affect the structural complexity and the understandability of the process flow. Hence, optional activities should be supported in BPMLs like BPMN, UML ADs, YAWL, and EPCs.

We therefore proposed to explicitly support optional activities in business process modelling. Based on the Workflow Patterns we identified optional activities in BPMLs by considering the Control-Flow Patterns, studying in detail the exclusive and multi-choice pattern. We further compared the semantics of diagrams with and without optional activities using ASMs. Then we applied our concept to BPMN and suggested three possible representations for optional activities, i.e. specifying an additional marker, defining a different background colour, or using a dashed line. Finally, we evaluated the proposed approach by introducing a graph transformation system and specifying several transformation rules to transform models with an implicit expression of modality to models with an explicit representation. The transformation always leads to equally many or fewer gateways and sequence flows and, thus, confirms a reduction of structural complexity. Furthermore, we showed within a case study that the use of explicitly optional activities not only reduces the structural complexity but also increases the understandability of a process flow.

In future work, we plan to investigate a full set of deontic classifications of activities, including obligation, permission, prohibition, and conditional commitments. Another issue is related to the limited support for actor modelling in BPMLs like BPMN. In this respect, we are currently developing a new approach including deontic logic and speech act theory that will be evaluated based on the resource perspective of the Workflow Patterns.

Acknowledgement. The project *Vertical Model Integration* is supported within the program "Regionale Wettbewerbsfähigkeit OÖ 2007-2013" by the European Fund for Regional Development as well as the State of Upper Austria. This work was further supported in part by the Austrian Science Fund (FWF) under grant no. TRP 223-N23.

References

[BS03] Egon Börger and Robert Stärk. *Abstract State Machines - A Method for High-Level System Design and Analysis*. Springer Verlag, 2003.

[CK07] Soo Ho Chang and Soo Dong Kim. A Service-Oriented Analysis and Design Approach to Developing Adaptable Services. In *IEEE SCC*, pages 204–211. IEEE Computer Society, 2007.

[KR09] Vera Künzle and Manfred Reichert. Towards Object-aware Process Management Systems: Issues, Challenges, Benefits. In *Proc. 10^{th} Int'l Workshop on Business*

Process Modeling, Development, and Support (BPMDS'09), number 29 in LNBIP, pages 197–210. Springer, 2009.

[KR11] Vera Künzle and Manfred Reichert. PHILharmonicFlows: towards a framework for object-aware process management. *Journal of Software Maintenance*, 23(4):205–244, 2011.

[MA02] Dirk Muthig and Colin Atkinson. Model-Driven Product Line Architectures. In Gary Chastek, editor, *Software Product Lines*, volume 2379 of *LNCS*, pages 79–90. Springer Berlin / Heidelberg, 2002.

[Mic10] Jutta Michely. Help with BPMN notation. `http://www.ariscommunity.com/users/johnm/2010-06-09-help-bpmn-notation`, 2010.

[Nat11] Christine Natschläger. Deontic BPMN. In A. Hameurlain, S. Liddle, K.-D. Schewe, and Xiaofang Zhou, editors, *Database and Expert Systems Applications*, volume 6861 of *LNCS*, pages 264–278. Springer Berlin / Heidelberg, 2011.

[Nat12] Christine Natschläger. *Extending BPMN with Deontic Logic*. PhD thesis, Johannes Kepler Universität Linz (JKU), 2012.

[Oes02] Bernd Oestereich. *Developing Software with UML: Object-Oriented Analysis and Design in Practice*. Pearson Education Ltd, 2 edition, 2002.

[OMG11] OMG. Business Process Model and Notation (BPMN) 2.0. `http://www.omg.org/spec/BPMN/2.0`, 2011.

[PSWW05] Frank Puhlmann, Arnd Schnieders, Jens Weiland, and Mathias Weske. Variability Mechanisms for Process Models. Technical report, 2005.

[RBSS05] Iris Reinhartz-Berger, Pnina Soffer, and Arnon Sturm. A Domain Engineering Approach to Specifying and Applying Reference Models. In *Proc. of the Work. Enterprise Modelling and Information Systems Architectures*, volume 75 of *Lecture Notes in Informatics*, pages 50–63. German Informatics Society, 2005.

[RK08] Maryam Razavian and Ramtin Khosravi. Modeling Variability in Business Process Models Using UML. In *Proc. of the 5^{th} Int. Conf. on Information Technology: New Generations*, ITNG '08, pages 82–87. IEEE Computer Society, 2008.

[Ros09] Marcello La Rosa. *Managing variability in process-aware information systems*. PhD thesis, Queensland University of Technology, 2009.

[Sil09] Bruce Silver. BPMN and Case Management. `http://www.bpmncase.com`, 2009.

[SP06] Arnd Schnieders and Frank Puhlmann. Variability Mechanisms in E-Business Process Families. In *Proc. Int. Conf. on Business Information Systems (BIS 2006)*, pages 583–601, 2006.

[tHvdAAR10] Arthur H.M. ter Hofstede, Wil M.P. van der Aalst, Michael Adamns, and Nick Russell, editors. *Modern Business Process Automation: YAWL and its Support Environment*. Springer, 2010.

[vdAtH11] Wil M.P. van der Aalst and Arthur H.M. ter Hofstede. Workflow Patterns Homepage. `http://www.workflowpatterns.com`, 2011.

[vdAtHKB03] Wil M.P. van der Aalst, Arthur H.M. ter Hofstede, Bartek Kiepuszewski, and Alistair P. Barros. Workflow Patterns. *Distributed and Parallel Databases*, 14:5–51, 2003.