Pattern-based Analysis of BPMN - an extensive evaluation of the Control-flow, the Data and the Resource Perspectives[⋆]

(Revised Version)

Petia Wohed[1⋆⋆], Wil M.P. van der Aalst[2,3], Marlon Dumas[3]
Arthur H.M. ter Hofstede[3], Nick Russell[3]

[1] The Department of Computer and Systems Sciences, Stockholm University/KTH
Forum 100, 164 40 Kista, Sweden
petia@dsv.su.se

[2] Faculty of Information Technology, Queensland University of Technology
GPO Box 2434, Brisbane QLD 4001, Australia
{m.dumas, a.terhofstede, n.russell}@qut.edu.au

[3] Department of Technology Management, Eindhoven University of Technology
GPO Box 513, NL5600 MB Eindhoven, The Netherlands
w.m.p.v.d.aalst@tm.tue.nl

Abstract. In this paper an evaluation of BPMN is presented, using the Workflow Patterns as an analysis framework. The analysis provided for BPMN is part of a larger effort aiming at an unbiased and vendor-independent survey of the expressive power of some mainstream modelling languages for process-aware information systems. It is a sequel to an analysis series where languages like BPEL and UML 2.0 A.D were evaluated. The results from the survey could both be used for the selection of a modelling technique, as well as for motivation and input to further development of any of the surveyed languages.
Keywords: BPMN, Workflow Patterns, Evaluation

1 Introduction

The shift from data to process orientation and the focus on process-aware information systems (PAIS) during the last decade, is closely tied to the development of a new generation of languages and tools for process description. Existing (mainstream) languages for IS development have recently been revised, e.g. UML2.0 was released with major updates on the behavioural part, i.e. UML Activity Diagrams (AD). In parallel, new languages like BPMN and BPEL4WS have been developed and have rapidly spread.

The common feature of these three languages is their focus on providing a powerful and standardized notation for representing the behavioural aspects of an enterprise.

[⋆] This work is funded in part by the Swedish Research Council under the grant "Analysis Patterns - An Approach for Flexible and Reusable Information Systems" and by the Australian Research Council under the Discovery Grant "Expressiveness Comparison and Interchange Facilitation Between Business Process Execution Languages".

[⋆⋆] Research conducted during a visit at the Queensland University of Technology.

Among the differences it can be mentioned that while UML AD and BPMN are graphical but not-formalised notations, BPEL4WS is an executable language (and therefore, in principle, also formalized) which lacks a graphical notation.

These rough characteristics do not, however, provide any insights into the languages' expressive capacity and into how they actually relate to each other. While some perceive UML2.0 AD and BPMN as potential graphical notations for BPEL4WS and hence as competitors [10], according to the following quotation,

> *"Where BPMN has a focus on business processes, the UML has a focus on software design and therefore the two are **not competing** notations but are different views on systems."* [3]

the proponents of BPMN, namely BPMI.org, obviously do not share this opinion. In order to be able to support or discard such statements, and more importantly in order to investigate the similarities and differences between the languages, a thorough analysis and comparison is necessary. This is the focus of the work partially presented here. The goal is to provide an unbiased survey of the expressive power of the process modelling languages and notations available today. For achieving this a number of languages have been analysed through one and the same analysis framework. This paper reports the results from the analysis of BPMN. This analysis is unbiased and the results differ from the results of a similar evaluation provided by White [19] (who is one of BPMN's developers). It is a sequel of reports in which corresponding evaluations of UML 2.0 AD and BPEL4WS were presented. It also demonstrates how, based on these evaluations, the languages can be compared and more precise conclusions about their similarities and differences can be drawn.

The Workflow Patterns (`www.workflowpatterns.com`) framework provides a reference analysis framework. This framework consists of a number of patterns which provide a taxonomy of generic, recurring concepts and constructs relevant in the context of process-aware information systems (PAIS). In accordance with Jablonski and Bussler's original classification [6], this framework spans the control-flow, data, and resource perspectives and was gradually developed to incorporate 20 *Control-flow patterns* [2], 40 *Data patterns* [15], and 43 *Resource patterns* [13]. Systematically investigating the possibility of expressing every pattern from the framework in a selected language, builds up a comprehensive picture of the scope and suitability of this language with respect to the three dimensions outlined above.

Our choice of the Workflow Patterns framework as an analysis framework is motivated by several factors. First, it is a well accepted framework which has been widely used both for the selection of workflow management systems (e.g., by UWV, the organization executing all regulations with respect to health and unemployment insurance in the Netherlands, the Dutch Justice Department, ArboNed, etc.) as well as for vendors' self-evaluations of products (e.g., COSA, FLOWer, Staffware, IBM, etc.)[1]. Second, this framework has proven impact in the industry. It has triggered extensions to workflow management systems (e.g., FLOWer 3.0, Staffware Process Suite, Pectra Technology Inc's tool) and inspired their development (e.g., OpenWFE, Zebra, Alphaflow). Third,

[1] For references on the impact of the Workflow Patterns initiative, please, refer to `www.workflowpatterns.com`.

this framework is at a sufficiently detailed level of abstraction to provide an instrument for assessing the capabilities of business process modelling languages. In contrast to the Bunge, Wand and Weber's (BWW) ontology [16], which is another framework widely used for evaluation of information systems' modelling languages, the Workflow Patterns framework was specifically tailored for the purposes of process language analysis. While BWW would be classified according to [5] as a top-level ontology for information systems, the Workflow Patterns framework would be classified as a *fine-grained* top-level ontology for process aware information systems. Fourth, as the Workflow Patterns framework spans the Control-flow, the Data and the Resource perspectives, it constitutes the most comprehensive framework currently in existence. The contributions of this paper are:

– Identification of limitations in BPMN, hence providing input for further development of the language.
– Discussions on how to capture the patterns in BPMN which provide elements of reusable knowledge for process designers that encounter these patterns.
– Outline of the differences from the evaluation provided by White [19] and discussion of the problems identified in White's evaluation[2].
– Identification of ambiguities in the current version of the specification [18].
– Results that can be used for a direct comparison of the expressive capacity of BPMN with other languages which have undergone a corresponding analysis, e.g., UML 2.0 AD, BPEL4WS, etc.

In the remainder of the paper we evaluate each of the three perspectives, starting by the Control-flow perspective, and followed by the Data and Resource perspectives. Then we discuss our findings and also compare these with earlier evaluations of UML 2.0 AD and BPEL.

2 Control-flow Patterns in BPMN

This section provides an analysis of BPMN in terms of the Control-flow patterns as defined in [2]. The main symbols of BPMN are summarized in Figure 1. For a description of BPMN the reader is referred to the specification [18]. All page references in the remainder of this paper refer to [18].

The Control-flow patterns are divided into five categories: Basic control-flow patterns; Advanced branching and synchronisation patterns; Structural patterns; Multiple instances patterns; State-based patterns; and Cancellation patterns.

2.1 Basic Control-flow Patterns

The basic Control-flow patterns define elementary aspects of process control. These are:

– CP1: *Sequence* – the ability to depict a sequence of activities;

[2] As the evaluation provided by White only covers the Control-flow patterns, this discussion is limited to the Control-flow perspective only.

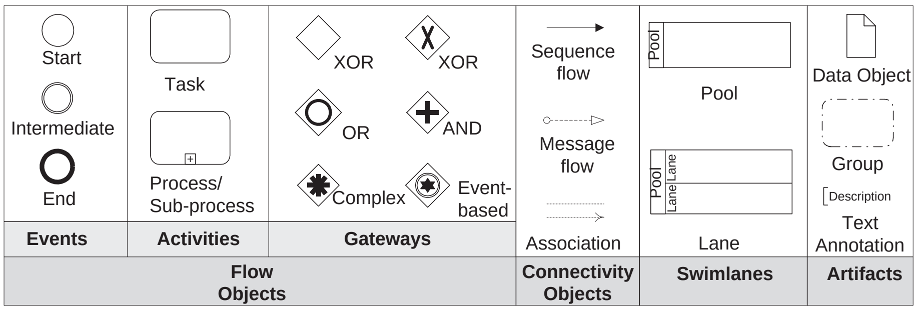

Fig. 1. BPMN, main symbols

- CP2: *Parallel split* – a split of a single thread of control into multiple threads of control which can execute concurrently;
- CP3: *Synchronisation* – a convergence of multiple parallel sub-processes/activities into a single thread of control thus synchronising multiple threads;
- CP4: *Exclusive choice* – a decision point in a workflow process where one of several branches is chosen;
- CP5: *Simple merge* – a point in the workflow process where two or more alternative branches come together without synchronisation.

These patterns correspond to control-flow constructs defined by the Workflow Management Coalition [17] and they are supported by basically all process modelling languages. For the sake of completeness and in order to outline the multiple ways of capturing them, we briefly discuss their solutions in BPMN.

The CP1 **Sequence** is represented through a Sequence Flow between two Activities. The CP2 **Parallel Split** is captured by an AND-split Gateway (see Figure 2a). Furthermore, the parallel split can also be modelled implicitly, by drawing the flows directly from the action node and omitting the AND-gateway (see Figure 2b). The Expression Type attribute of *b1* and *b2* have to be set to *None* (which is the default setting). A third solution is to model the parallel Activities as Sub-Activities in a Process/Sub-Process (see Figure 2c). This solution is inspired by Figure 10 on p. 68 in [18].

The CP3 **Synchronisation** is captured through an AND-join Gateway (see Figure 2d). Here too, a solution using Sub-Activities is offered (see Figure 2e which is inspired by Figure 10 in [18]). However, this solution cannot be used in some complex situations, for instance, the one depicted in Figure 2f, where only one of the subsequent tasks requires synchronisation. For a detailed discussion on this topic we refer to [7].

The CP4 **Exclusive Choice** is captured through the XOR-split Gateway (see figures 2g and 2h). Optionally, one of the gates may be *default*. If used, the semantics of the construct guarantees that no matter what conditions are specified, exactly one outgoing flow will be chosen. Furthermore, the behaviour of the Exclusive Choice pattern can be achieved through the use of the attributes ConditionType and ConditionExpression of the Flows with the same source Activity. This is done by setting the value of the ConditionType attributes to *Expression* and specifying mutually exclusive expressions for the ConditionExpression attributes (see Figure 2i). Here too, it is up to the designer

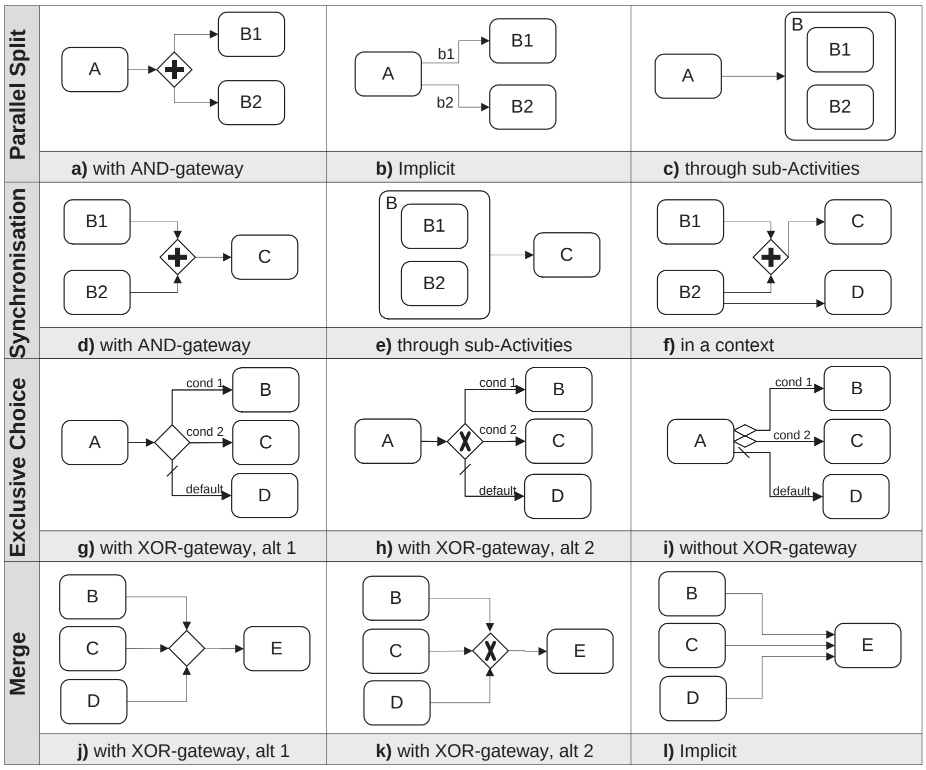

Fig. 2. Basic Control-flow patterns in BPMN

to guarantee the totality of the set of the expressions (i.e., by for instance the use of the *default* flow).

The CP5 **Simple Merge** pattern is expressed by using the XOR-join Gateway (see figures 2j and 2k). Furthermore, supported by the statements "If the Sub-Process/Task has multiple incoming Sequence Flow" and "... when a Token arrives from one of the Paths, the Sub-Process/Task will be initiated. [..] If another Token arrives from the same path or another path, then a separate instance of the Sub-Process/Task will be created" (p. 73 and p. 81) the construct in Figure 2l also provides a solution for the Simple Merge pattern.

Note that, according to p. 86 in [18] the behaviour of the models in Figure 2j and Figure 2l "are the same [only] if all the incoming flow are alternative". Any further explanation of the restriction on alternative flows is, however, not provided.

2.2 Advanced Branching and Synchronisation Patterns

This class of patterns corresponds to advanced branching and synchronisation scenarios relatively common in real-life business processes. There are four of these patterns:

– CP6: *Multiple Choice* – the ability to represent a divergence of the thread of control into several parallel branches on a selective basis;

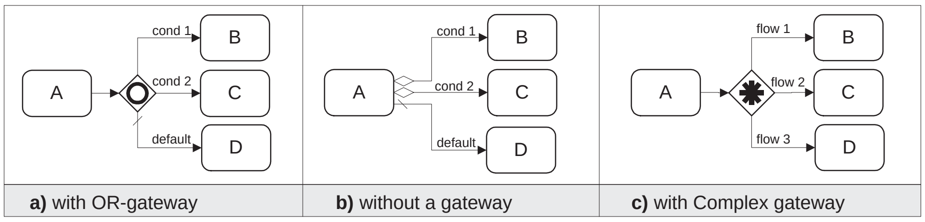

Fig. 3. Multiple Choice in BPMN

- CP7: *Synchronising merge*– the ability to depict the synchronised convergence of two or more alternative branches;
- CP8: *Multiple Merge* – the ability to represent the unsynchronised convergence of two or more distinct branches. If more than one branch is active, the activity following the merge is started for every activation of every incoming branch;
- CP9: *Discriminator*– the ability to depict the convergence of two or more branches such that the first activation of an incoming branch results in the subsequent activity being triggered and subsequent activations of remaining incoming branches are ignored. The discriminator is a special case of the N-out-of-M Join where N=1.

In the CP6 **Multiple Choice** pattern, in contrast to the Exclusive Choice pattern, zero, one or multiple outgoing branches may be chosen. In BPMN there are three ways for capturing this pattern: i) With an OR-split Gateway (see Figure 3a). "It is up to the modeller to insure that at least one of the conditions will be TRUE" (p. 95), which for instance can be achieved through the use of the *default* flow; ii) without any Gateway construct (see Figure 3b). The difference from the solution in Figure 2i is that the Condition Expressions for the different flows are not exclusive; and iii) with a Complex Gateway (see Figure 3c) where an expression specifies which set of flows will be executed in different situations.

The CP7 **Synchronising Merge** pattern is captured partially through the OR-join Gateway. The solution proposed by White in [19, 18] and redrawn in Figure 4a assumes a structured workflow context. If the pattern appears in an unstructured workflow, for instance the one depicted in Figure 4b[3], the OR-join Gateway will not capture the desired behaviour. Consider for instance the following scenario. After the completion of

[3] The scenario in this figure is taken from [4].

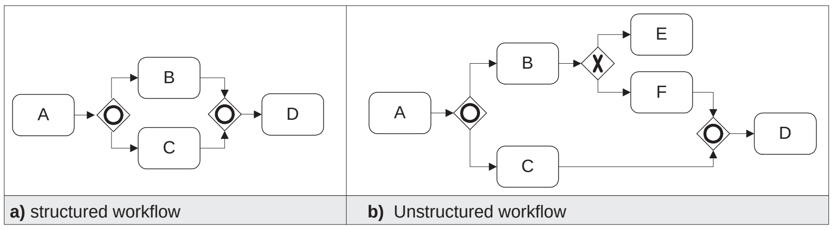

Fig. 4. Synchronising Merge in BPMN

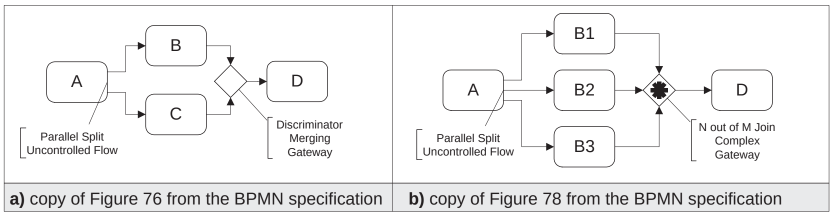
A
B
C
D
Parallel Split Uncontrolled Flow
Discriminator Merging Gateway
a) copy of Figure 76 from the BPMN specification
A
B1
B2
B3
D
Parallel Split Uncontrolled Flow
N out of M Join Complex Gateway
b) copy of Figure 78 from the BPMN specification

Fig. 5. Solutions proposed in the BPMN specification for the Discriminator pattern

activity A, both activity B and activity C are initiated. After completion of B, activity E is selected and triggered. According to the semantics of the Synchronising Merge pattern, after the completion of C, activity D should be enabled. This is because the workflow will no longer be able to reach a state where the branch incoming to the merge from activity F will be completed. However, according to the semantics of the OR-join Gateway, activity D will not be enabled because the OR-join gateway "will wait for [synchronising] all Tokens that have been produced upstream"(p. 95). The flow will deadlock because the token in E (produced upstream) is not merging into the OR-join Gateway expecting it. For a more complete treatment of OR-joins see [22, 8].

The CP8 **Multiple Merge** pattern is expressed in the same way as the CP5 Simple Merge pattern (see figures 2j, 2k and 2l).

The CP9 **Discriminator** pattern is a special case of the N-out-of-M Join pattern, where N=1. N-out-of-M Join depicts the ability of synchronising a flow after N parallel threads (out of M initiated threads) have completed. The solutions proposed by White for the Discriminator and the N-out-of-M Join patterns are reprinted in figures 5a and 5b (cf. figures 76 and 78 in [18]), respectively. Note that both these solutions: (i) delimit the context in which the pattern is used by defining a parallel split preceding the tasks involved in the join; and (ii) they rely fully on textual annotations, i.e., not a single gateway or a split is left without a clarifying annotation. Furthermore, the textual description explaining the Discriminator in figure 76, contradicts the definition of the XOR-Gateway. E.g., the claim "In this situation [referring to Figure 76], the Gateway will accept the first Token and immediately pass it on through to the activity [D]. When the second Token arrives, it will be *excluded* from the remainder of the flow. This means that the Token will not be passed on to the activity [D], but will be consumed." (p. 133) contradicts the statement "If there are multiple incoming Sequence Flow, all of them will be used to continue the flow of the Process [..]. That is, Process flow SHALL continue when a signal (a Token) arrives from any of a set of Sequence Flow." (p. 89) in the definition of the XOR-Gateway. Because of this inconsistence, we discard the solution for the Discriminator (Figure 5a) proposed in [18] and suggest instead the solutions in Figure 6.

Figure 6a shows a partial solution for the Discriminator pattern, namely when the patterns is utilized in the context of a multiple instances task. The values *M* and *Parallel* for the MI_Condition and MI_Ordering attributes in the multiple instances task B indicate that M instances of B will be created and run in parallel. The value ***One*** in MI_FlowCondition attribute implies that "Token SHALL continue past the activity [B] after only one of the activity instances has completed" (p.65). The solution also works

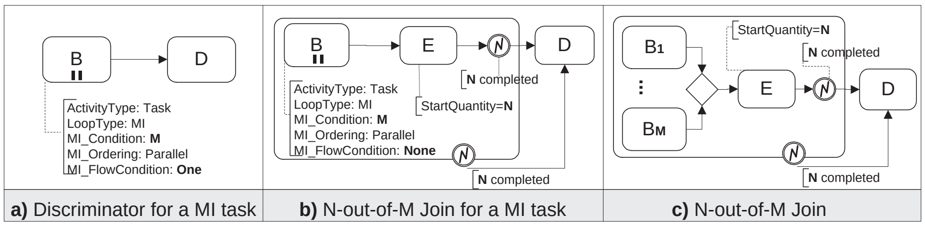

Fig. 6. Alternative Solutions for the Discriminator pattern

in the context of a loop, but is not generalisable to the N-out-of-M join pattern, as *One* is the only numerical value that can be chosen for this attribute. In order to capture the general case of N-out-of-M join the solution is extended as shown in Figure 6b. The MI_FlowCondition attribute is now set to *None*, which means that every instance will generate a token that will continue downstream when the instance is completed (p.65). Furthermore, an empty activity E with a StartQuantity attribute set to N is introduced to synchronise the flow by delaying the execution of E until N number of tokens are received[4]. In order to deal with the remaining of the instances, the completions of which are not longer relevant (so that the solution is general and works also for instance in the context of loops), this construct is placed in a sub-process, which is cancelled immediately after the execution of E. We assume that, even if there are two different outgoing flows, the process will allays continue through the flow outgoing from the Error Event "catching" the *N completed* signal. This solution assumes that an Error Event signal can be "thrown" from the same sub-process which "catches" it. A similar work-around for the case when the pattern is used in the context of distinct activities is proposed in Figure 6c. Note the (non-trivial) attribute setting in these solutions.

2.3 Structural Patterns

Structural patterns identify whether the modelling formalism has any restrictions in regard to the way in which processes can be structured (particularly in terms of whether loops are supported or whether a single terminating node is necessary). There are two of these patterns:

- CP10: *Arbitrary cycles* – the support for multiple ways of entering end exiting the areas with repetitive activities;
- CP11: *Implicit termination* – the ability to depict the notion that a given sub-process should be terminated when there are no remaining activities to be completed.

BPMN provides direct support for the CP10 **Arbitrary Cycles** (also pointed out by White [19]). Furthermore, BPMN also provides direct support for the CP11 **Implicit Termination** pattern which is captured by ending every thread within a Process with an End Event. An End Event completes either a thread of control, or if it consumes the last token generated by the Start Event of a process it completes the whole process.

[4] An alternative could be to use the attribute Quantity of the flow outgoing from task B. However, the semantics of this attribute is not entirely clear.

2.4 Multiple Instances Patterns

The Multiple instances (MI) patterns refer to situations where there can be more than one instance of a task active at the same time in the same case.

- CP12: *MI without Synchronisation* – the ability for one case to initiate multiple instances of an activity;
- CP13: *MI with a Priori Design Time Knowledge*– the ability for one case to initiate multiple instances of an activity and when completed to synchronise them. The number of instances is known at design time;
- CP14: *MI with a Priori Runtime Knowledge*– As the previous pattern, but the number of instances to be created is first known at runtime before the instances must be created;
- CP15: *MI without a Priori Runtime Knowledge*– As the previous pattern with the extension that the number of instances to be created is not known a priori and new instances can be created even while other instances are executing or have already completed.

A solution for the CP12 **MI without Synchronisation** is provided in Figure 7a. M number of instances of Task B will be "spawned-off" during execution. These instances will execute in parallel. The setting *None* for the MI_Flow Condition attribute, implies that the initiated instances will not be synchronised but after completion every instance of B will produce a Token which will continue downstream though the flow and trigger an execution of the subsequent task C.

A setting *All* for the MI_Flow Condition attribute (see Figure 7b) implies synchronisation of the created instances and provides a solution for the CP13 **MI with a Priori Design Time Knowledge** pattern. This solution is also applicable for the CP14 **MI with a Priori Runtime Knowledge** pattern, with the difference that MI_Condition is a variable the assignment of which is made during run-time.

The solution proposed by White in [19] utilizes the notion of MI Activities that are executed in *Sequence*. This means that evaluation of whether a new instance will be spawned-off is done every time an existing instance is completed. This lack of support for parallel execution of the multiple instances is considered as too strong a deviation from the semantics of the pattern.

The CP15 **MI without a Priori Runtime Knowledge** pattern is not directly supported in BPMN. In Figure 8a a workaround solution is proposed. In this solution B is the Activity for which multiple instances will be created. The number of instances to be

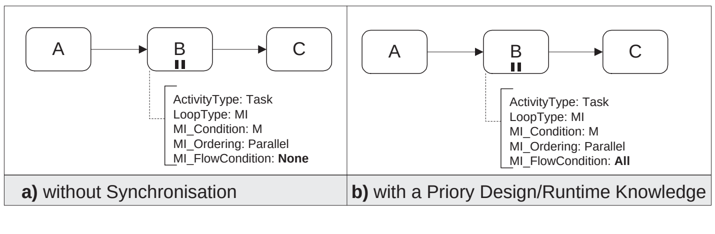

Fig. 7. Multiple Instances in BPMN

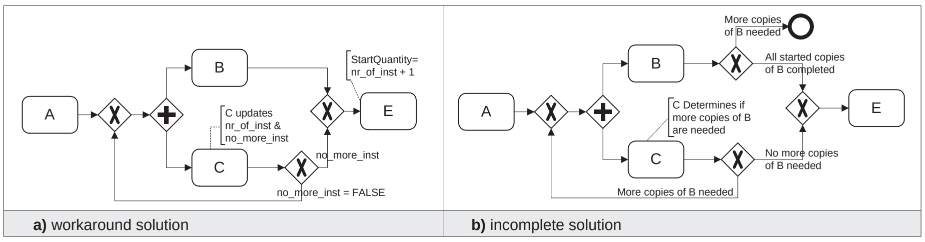
A
B
C
E
StartQuantity= nr_of_inst + 1
C updates nr_of_inst & no_more_inst
no_more_inst
no_more_inst = FALSE
a) workaround solution
More copies of B needed
All started copies of B completed
C Determines if more copies of B are needed
No more copies of B needed
More copies of B needed
b) incomplete solution

Fig. 8. MI without a Priori Runtime Knowledge

created is controlled through the activity C which is executing in a parallel flow to this of B. In C the value of the variable *no_more_inst* can be changed. If FALSE the flow loops back so that a new instance of B will be created. If TRUE, i.e., when no more instances of B are needed, the flow continues to a merging point (i.e. the XOR-join Gateway in the model). Furthermore, an additional variable *nr_of_inst* is introduced for keeping track of the number of created instances of B. This number is later on necessary for knowing how many instances there will be to synchronise. As the semantics of the XOR-join Gateway is that "Process Flow SHALL continue when a signal (a Token) arrives from any of a set of Sequence Flow." (p. 89), the XOR-join Gateway will merge, but not synchronise the completed instances. In order to synchronise them the attribute *StartQuantity* of Activity E needs to be set to *nr_of_inst + 1*, so that E will be triggered first when all created instances of B have completed (*nr_of_inst* plus one, because of the merging token from activity C).

The solution for this pattern proposed by White in [19] and reprinted in Figure 8b will deadlock in the situation when all necessary copies/instances of B are started and they are running in parallel. The moment the first copy/instance completes the XOR-join following B will deadlock because both Condition Expressions for its outgoing branches will evaluate to FALSE. Another question this solution raises is how the condition *All started copies of B completed* will be evaluated.

2.5 State-based patterns

This class of patterns characterises scenarios in a process where subsequent execution is determined by the state of the process instance. There are three such patterns:

- CP16: *Deferred Choice* – the ability to depict a divergence point in a process where one of several possible branches should be activated. The actual decision on which branch is activated is made by the environment and is deferred to the latest possible moment;
- CP17: *Interleaved Parallel Routing* – the ability to depict a set of activities that can be executed in arbitrary order;
- CP18: *Milestone* – the ability to depict that a specified activity cannot be commenced until some nominated state is reached.

As the notion of Event is separated from the notion of Activity, the CP16 **Deferred Choice** pattern is easily captured in BPMN. Three different solutions are presented

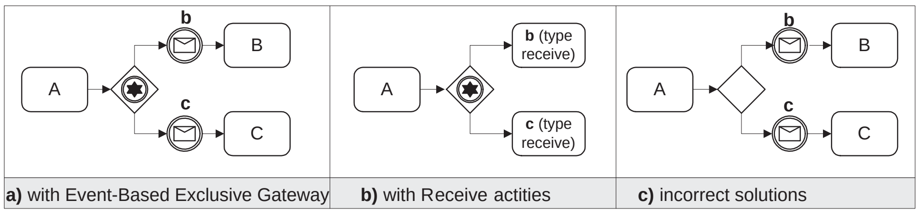

Fig. 9. Deferred Choice in BPMN

in [18]. The first one (see Figure 9a based on Figure 25 on p. 91 in [18]) utilizes the construct of Event-Based Exclusive Gateway with Intermediate Events with the Trigger attribute set to *Message*. ".. the basic idea is that this Decision [referring to the Event-Based Exclusive Gateway] represents a branching point in the process where the alternatives are based on events that occur at that point in the Process, rather than the evaluation of expressions using process data." (p. 90). The second solution (see Figure 9b which is based on Figure 24 on p. 90 in [18]) is similar to the first one, but instead of Message Events, Receive Tasks (i.e. Tasks with the Task Type attributes set to *Receive*) are used. Finally, the third solution utilizes the notion of XOR-Gateway in combination with Message Events (see Figure 9c, which is based on Figure 72 on p.131 in [18]). However, this solution is in conflict with the fact that every Sequence Flow outgoing from an XOR-split Gateway "MUST have its Condition attribute set to Expression and MUST have a valid Condition Expression." (p. 88).

For capturing the CP17 **Interleaved Parallel Routing** pattern White proposes the construct of an Ad Hoc Process consisting of a number of Sub-Processes [19] (see Figure 10a). The attribute AdHocOrdering has to be set to *Sequential*. Also an AdHocCompletionCondition expression (p. 70), specifying that the Process will complete when all its Sub-Processes have completed, has to be defined (these two attributes do not have any graphical representation).

This solution works as long as the activities to be interleaved are simple tasks (in the example these are the tasks A and C). If, instead, sequences of activities need to be interleaved, e.g. the sequence of activities A.B needs to be interleaved with the sequence of the activities C.D (which will allow any of the following execution sequences: A.B.C.D, A.C.B.D, A.C.D.B, C.D.A.B, C.A.B.D, and C.A.D.B), the semantics of the AdHoc Processes is not clear. If the solution in Figure 10b is used, the execution sequences will probably be delimited into the following two orderings A.B.C.D and C.D.A.B. To avoid

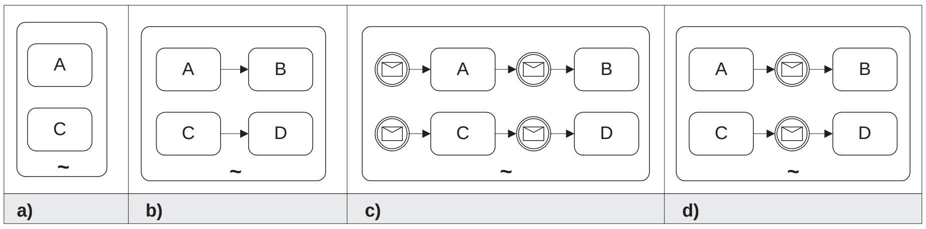

Fig. 10. Solutions for Interleaved Parallel Routing

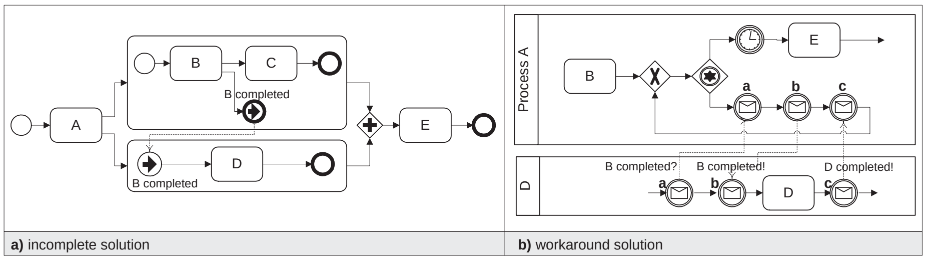

Fig. 11. Solutions for Milestone

this limitation, Intermediate Events of type *Message* can be included (see Figure 10c). These Events will simulate *States* between the Activities and allow an execution sequence where switching between the flows is possible, preserving at same time the sequencing within the flows. This solution assumes, however, that sequence flows can be defined within AdHoc Activities. It also raises the question whether the model in Figure 10d have the same behaviour as the model in Figure 10c.

The CP18 **Milestone** pattern is not easily captured because of the lack of support for the notion of states. The solution proposed by White in [19] and reprinted in Figure 11a does not model the expiration of the milestone. An activity (activity D in the example) which *potentially* can be executed at a certain milestone (in the example, after B has completed and before E has started) is *always* executed.

A workaround for this pattern is proposed in Figure 11b. In this solution activity D, which can potentially be run at the nominated milestone, is modelled in a separate process. A check (through a message exchange) whether the milestone is reached is performed every time before D is started. The milestone is modelled by an Event-Based Decision with two branches: one for capturing the expiration of the milestone and continuing the process flow; and another one for taking care of the communication and the inquiries on the "current state" of the process. A draw-back of this solution is that the milestone is re-initialised at every entry of the loop. If relative time is used, the milestone also needs to be re-calculated.

2.6 Cancellation Patterns

There are two cancellation patters:

- CP19: *Cancel activity* – the ability to depict that an enabled activity should be disabled in some nominated circumstance;
- CP20: *Cancel case* – the ability to represent the cancellation of an entire process instance (i.e. all activities relating to the process instance) in some nominated circumstance.

In BPMN, the CP19 **Cancel Activity** pattern can be captured as shown in Figure 12a. To achieve the desired behaviour, an Intermediate Event of type *Error* with ErrorCode attribute set to *Cancel* is attached to the boundary of the activity to be cancelled, i.e. activity A in the figure.

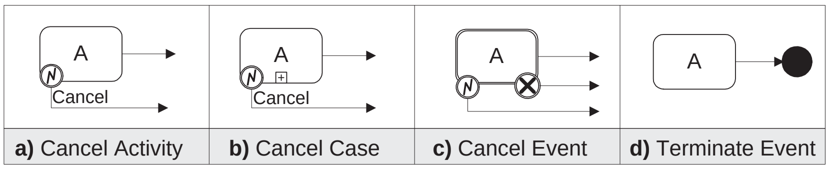

Fig. 12. Cancellation concepts

The same construct can also be used to capture the CP20 **Cancel Case** pattern, with the difference that the Activity with a Cancellation Error Event is a Sub-Process (see Figure 12b). As the BPMN specification [18] is not explicit about what happens to the token(s) already existing within the Sub-Process, we assume that they are all "consumed" in the moment an Error Event catches a *Cancel* error. The process proceeds through the outgoing flow from the Error Event.

Another way to capture cancellation is through the notion of Transaction (see Figure 12c). Ending the outgoing flow from the Error Event with an End Event gives solution for the Cancel Case pattern. Finally, the Cancel Case pattern can be captured through the Terminate End Event (see Figure 12d). "This type of End indicates that all activities in the process should be immediately ended." (p. 54). The Terminate End Event symbol is placed where cancellation should be signalled. This is in line with the solutions proposed by White [19].

Table 1 summarises the results from this part of the evaluation. Note that the table not only shows the evaluation of BPMN. It also shows the results for UML 2.0 Activity Diagrams (as evaluated in [21, 14]), BPEL4WS (cf. [20, 1]), and a concrete system based

	1	2	3	4
Basic Control–flow				
1. Sequence	+	+	+	+
2. Parallel Split	+	+	+	+
3. Synchronisation	+	+	+	+
4. Exclusive Choice	+	+	+	+
5. Simple Merge	+	+	+	+
Advanced Synchronisation				
6. Multiple Choice	+	+	+	+
7. Synchronising Merge	+/–	–	+	+
8. Multiple Merge	+	+	–	–
9. Discriminator	+/–	+	–	–
Structural Patterns				
10. Arbitrary Cycles	+	+	–	–
11. Implicit Termination	+	+	+	+
Multiple Instances Patterns				
12. MI without Synchronization	+	+	+	+
13. MI with a priori Design Time Knowledge	+	+	+	+
14. MI with a priori Runtime Knowledge	+	+	–	+
15. MI without a priori Runtime Knowledge	–	–	–	+/–
State-Based Patterns				
16. Deferred Choice	+	+	+	+
17. Interleaved Parallel Routing	+/–	–	+/–	–
18. Milestone	–	–	–	+/–
Cancellation Patterns				
19. Cancel Activity	+	+	+	+/–
20. Cancel Case	+	+	+	+

Table 1. Support for the Control–flow Patterns
in **1**–BPMN, **2**–UML2.0 AD [21, 14], **3**–BPEL [20, 1], and **4**–Oracle BPEL PM v.10.1.2 [11]

on the latter language Oracle BPEL Process Manager (PM) Version 10.1.2 [11][5]. In the conclusion we will compare BPMN with these other languages.

3 Data Patterns in BPMN

The Data perspective focuses on identifying and defining generic constructs for data representation and handling within process aware information systems [15]. In total 40 data patterns have been delineated in four distinct groups – data visibility, data interaction, data transfer and data-based routing.

3.1 Data visibility patterns

Data visibility patterns characterise the various ways in which data elements can be defined and utilised within the context of a process. In general, this is determined by the main construct to which the data element is bound as it implies a particular scope in which the data element is visible and capable of being utilised. There are eight patterns which relate to data visibility:

- DP1: *Task data* – data elements defined and accessible in the context of individual execution instances of a task or activity;
- DP2: *Block data* – data elements defined by block tasks (i.e. tasks which can be described in terms of a corresponding decomposition) and accessible to the block task and all corresponding components within the associated decomposition;
- DP3: *Scope data* – data elements bound to a subset of the tasks in a process instance;
- DP4: *Multiple instance data* – data elements specific to a single execution instance of a task (where the task is able to be executed multiple times);
- DP5: *Case data* – data elements specific to a process instance which are accessible to all components of the process instance during execution;
- DP6: *Folder data* – data elements bound to a subset of the tasks in a process definition but accessible to all task instances regardless of the case to which they correspond;
- DP7: *Workflow data* – data elements accessible to all components in all cases;
- DP8: *Environment data* – data elements defined in the operational environment which can be accessed by process elements.

BPMN supports several of these patterns. The smallest operational unit in a BPMN diagrams is Task. Task data is defined through the attribute Properties of a Task. These properties are local, hence they are only for use within the Task (p. 63).

Sub-Processes and Processes serve as the main grouping mechanism in BPMN and they have similar characteristics to the block construct and the concept of case in process definitions. The Block Data Pattern is directly supported through the attribute

[5] Note that there are some minor difference between the BPEL standard and the way it is implemented in Oracle BPEL PM. Moreover, unlike the standard, support for the Resource perspective is given (i.e., work distribution, user tasks, etc.).

Properties of a Sub-Processes (p.63) which are local and accessible to all Sub-Process components. Similarly the Case Data Pattern is directly supported through the attribute Properties of a Process (p.43).

Scope data is not supported. The Group construct is purely used for visualization purposes (p. 112) and it does not provide any data handling for the objects it groups together.

Multiple instance data is only partially supported. There are three situations where multiple instances of a given task may arise:

1. Where a task is specifically designated as having multiple instances in the process model. The lack of any data attributes in Table 18, p.65 [18] makes that it is not possible to handle any instance specific data for the different instances of a multiple instances task.
2. Where a task can be triggered multiple times, e.g., it is part of a loop or it is a task following after a multiple merge construct. These situations are allowable in BPMN.
3. Where two tasks share the same decomposition. This is also supported in BPMN. An activity decomposition can be captured through the notion of an Independent Sub-Process. An Independent Sup-Process is an activity in a process diagram which invokes another Process within the process diagram. Several Independent Sub-Processes (i.e. activities) can invoke the same Process (p.70).

Folder, Workflow and Environment data patterns are not supported in BPMN.

3.2 Data interaction patterns

Data interaction patterns deal with the various ways in which data elements can be passed between components within a process instance and also with the operating environment (e.g., data transfer between a component of a process and an application, data store or interface that is external to the process). They examine how the characteristics of the individual components can influence the manner in which the trafficking of data elements occurs. There are six internal data interaction patterns:

– DP9: *Data elements flowing between task instances*;
– DP10: *Data elements flowing to a block*;
– DP11: *Data elements flowing from a block*;
– DP12: *Data elements flowing to a multiple instance task instance*;
– DP13: *Data elements flowing from a multiple instance task instance*;
– DP14: *Data elements flowing between process instances or cases*.

Data interaction between tasks can be utilized in three different ways, namely: (i) through integrated control and data channels or; (ii) through distinct control and data channels or; (iii) through the support of global shared data. As BPMN supports global shared data (through the Properties attribute for a Process, p. 43) the third alternative is clearly supported. It also appears that the first two alternatives are supported. Data interaction through distinct control and data channels is supported through the notion of Data Object (p. 108) with its Properties attribute (see table 41 on p. 110 and fig. 40

on p. 109). Data interaction through integrated control and data channels is supported through the construct of Data Objects associated to Sequence Flows (see fig. 39 on p. 109).

Furthermore, both the data interactions task to sub-workflow and sub-workflow to task (DP10 and DP11) are directly supported. Three ways of doing this are possible: (i) implicit data passing, (ii) explicit data passing via parameters, and (iii) explicit data passing via channels. BPMN supports the first two of these alternatives. In the cases when a decomposition is defined through an Embedded Sup-Process (see p. 69 and p. 43), the data passing to and from the Sub-Process are realised implicitly, i.e. through global shared data. In the cases when a decomposition is defined through an Independent Sub-Process (p.70), the data transfer is realised via parameters, i.e., it is defined through the Input- and OutputPropertyMaps Expressions for the Sub-Process (see table 21 on p.71).

The Multiple Instances Activities constructs (pp. 63–65) allow nominated activities of a process model to be executed multiple times in sequence (provided the MI_Ordering attribute is set to *Sequential*) or in parallel (provided the MI_Ordering attribute is set to *Parallel*). Data passing into and out of a Multiple Instance Activity is done either implicitly, through general shared data, or through the Input- and OutputSets defined for this activity. As any instance specific data can not be explicitly defined (see DP4), the only possibility to handle such data would be to use Input and OutputSets as lists containing instance specific data and to use the LoopCounter for indexing these lists so that every instance gets its "own" data space. The instance specific data and the general task data would in this case be divided into different Sets. This workaround solution is, however, partial as it is only applicable to multiple instances executed sequentially (MI activities executed in parallel does not have any LoopCounter attribute). Furthermore, it is not clear how aggregation of instance specific data can be achieved. IORules which can be used to "define the relationship between one InputSet and one OutputSet." (Table 16, p. 64) are too limited to be used in the situation when the data from one OutputSet needs to be aggregated and the aggregation itself is part of the activity's general output.

The data interaction – case to case pattern (DP14) is not supported in BPMN.

In addition to the internal data interaction patterns, there are 12 external data interaction patterns. These are characterised by three dimensions:

- The type of process element – task, case or complete process – that is interacting with the environment;
- Whether the interaction is push or pull-based;
- Whether the interaction is initiated by the process element or the environment.

The Patterns Task to Environment, Push and Pull, and Environment to Task, Push and Pull, (i.e., DP 15, 16, 17 and 18) are supported in BPMN. They are captured through one or a pair of Message Flow(s) flowing to, from, or to and from a Task and the boundary of a Pool representing the Environment. Note that for these patterns the environment is modelled explicitly.

The patterns Case to Environment, Push and Pull, as well as Environment to Case, Push and Pull (i.e., the DP 19–22) are not supported. Message Flows can indeed be drawn between the boundaries of two Pools (p. 116) where one of the Pools represents a Process and the other one the Environment. However, "If the Message Flow is connected to the boundary to the Expanded Sub-Process, then this is equivalent to connecting to the Start Event for incoming Message Flow or the End Event for outgoing Message Flow." (p. 117). Hence, this construct does not provide support for data exchange of case data at *any* moment during the execution of a case.

Finally, the Workflow to Environment, Push and Pull, and Environment to Workflow, Push and Pull patterns, (i.e., DP 23-26), are not supported, as workflow data is not supported in BPMN (see DP 7 above).

3.3 Data transfer patterns

Data transfer patterns focus on the way in which data elements are actually transferred between one process element and another. They aim to capture the various mechanisms by which data elements can be passed across the interface of a process element. There are seven distinct patterns in this category:

- DP27: *Data transfer by value – incoming* –incoming data elements passed by value;
- DP28: *Data transfer by value – outgoing* – outgoing data elements passed by value;
- DP29: *Data transfer – copy in/copy out* – where a process element synchronises data elements with an external data source at commencement and completion;
- DP30: *Data transfer by reference – without lock* – data elements are communicated between components via a reference to a data element in some mutually accessible location. No concurrency restrictions are implied;
- DP31: *Data transfer by reference – with lock* – similar to DP30 except that concurrency restrictions are implied with the receiving component receiving the privilege of read-only or dedicated access to the data element;
- DP32: *Data transformation – input* – where a transformation function is applied to a data element prior to it being passed to a subsequent component;
- DP33: *Data transformation – output* – where a transformation function is applied to a data element prior to it being passed from a previous component.

In BPMN, the DP 27 and 28 patterns are supported through the notion of the Input and OutputSets (p.63). The DP 29 Data transfer copy in/copy out is partially supported. It occurs when a decomposition is realised with Independent Sub-Processes. The data attributes to be copied into/out of the Independent Sub-Process are specified through the Input- and OutputPropertyMaps attributes (p. 71). As these PropertyMaps are in the form of Expressions we assume that also different transformation functions can be captured through them. Transformation functions can also be defined through Expression Assignments on Gates. These implies that patterns DP32 and DP33 are partially supported as well. The support is considered to be partial because it only applies to data transfer to and from Independent Sub-Processes or to Activities subsequent to a Gateway, and not between any couple of Activities.

Finally, the DP 31 data transfer by reference – with lock is supported. As BPMN adopts a token-oriented approach to data passing, the parameters – which typically relate to objects – are effectively consumed at activity commencement and only become visible and accessible to other activities once the specific activity to which they were passed has completed and returned them.

3.4 Data-based routing patterns

Data-based routing patterns capture the various ways in which data elements can interact with other perspectives and influence the overall execution of the process. There are seven (relatively self-explanatory) patterns in this category:

- DP34: *Task precondition – data existence*;
- DP35: *Task precondition – data value*;
- DP36: *Task postcondition – data existence*;
- DP37: *Task postcondition – data value*;
- DP38: *Event-based task trigger*;
- DP39: *Data-based task trigger*;
- DP40: *Data-based routing*.

BPMN does not directly support pre- and postcondition definitions. Hence, the patterns 35 and 37 are not supported. In the cases data transfer is realised though Data Objects, the boolean attributes RequiredForStart and ProducedAtCompletion (p. 110) capture the pre- and postconditions for data existence (i.e., patterns 34 and 36).

The Message, Timer, Error and Cancel Event constructs (see Table 14 on p.58) provide direct support for the event-based task triggering pattern. The Rule Event construct (Table 14, p. 58) provides support for the Data-based task trigger pattern. Finally, the Data-based routing is supported, as Condition Expressions are possible to specify for Sequence Flows (see Table 45 on p. 115).

Table 2 shows a summary of the results from the Data perspective.

4 Resource Patterns in BPMN

The Resource perspective focuses on the manner in which work is distributed amongst and managed by the resources in a process-aware information system (PAIS). Forty three workflow resource patterns are identified in [13] and classified into seven distinct groups:

- *Creation patterns* – which correspond to restrictions on the manner in which specific work items can be advertised, allocated and executed by resources;
- *Push patterns* – which describe situations where a PAIS proactively offers or allocates work to resources;
- *Pull patterns* – which characterise scenarios where resources initiate the identification of work that they are able to undertake and commit to its execution;
- *Detour patterns* – which describe deviations from the normal sequence of state transitions associated with a business process either at the instigation of a resource or the PAIS;
- *Auto-start patterns* – which relate to situations where the execution of work is triggered by specific events or state transitions in the business process;
- *Visibility patterns* – which describe the ability of resources to view the status of work within the PAIS;

Data Visibility	1	2	3	4	Data Interaction (External) (cont.)	1	2	3	4
1. Task Data	+	+/–	+/–	+/–	21. Env. to Case – Push-Oriented	–	–	–	–
2. Block Data	+	+	–	–	22. Case to Env. – Pull-Oriented	–	–	–	–
3. Scope Data	–	–	+	+	23. Workflow to Env. – Push-Oriented	–	–	–	–
4. Multiple Instance Data	+/–	+	–	+/–	24. Env. to Workflow – Pull-Oriented	–	–	–	–
5. Case Data	+	–	+	+	25. Env. to Workflow – Push-Oriented	–	–	–	–
6. Folder Data	–	–	–	–	26. Workflow to Env. – Pull-Oriented	–	–	–	–
7. Workflow Data	–	+	–	–	Data Transfer				
8. Environment Data	–	–	+	+	27. by Value – Incoming	+	–	+	+
Data Interaction (Internal)					28. by Value – Outgoing	+	–	+	+
9. between Tasks	+	+	+	+	29. Copy In/Copy Out	+/–	–	–	+
10. Block Task to Sub-wf Decomp.	+	+	–	–	30. by Reference – Unlocked	–	–	+	+
11. Sub-wf Decomp. to Block Task	+	+	–	–	31. by Reference – Locked	+	+	+/–	–
12. to Multiple Instance Task	–	+	–	+/–	32. Data Transformation – Input	+/–	+	–	–
13. from Multiple Instance Task	–	+	–	+/–	33. Data Transformation – Output	+/–	+	–	–
14. Case to Case	–	–	+/–	–	Data-based Routing				
Data Interaction (External)					34. Task Precondition – Data Exist.	+	+	+/–	–
15. Task to Env. – Push-Oriented	+	–	+	+	35. Task Precondition – Data Val.	–	+	+	+
16. Env. to Task – Pull-Oriented	+	–	+	+	36. Task Postcondition – Data Exist.	+	+	–	–
17. Env. to Task – Push-Oriented	+	–	+/–	+	37. Task Postcondition – Data Val.	–	+	–	–
18. Task to Env. – Pull-Oriented	+	–	+/–	+	38. Event-based Task Trigger	+	+	+	+
19. Case to Env. – Push-Oriented	–	–	–	–	39. Data-based Task Trigger	+	–	+/–	–
20. Env. to Case – Pull-Oriented	–	–	–	–	40. Data-based Routing	+	+	+	+

Table 2. Support for the Data Patterns
in **1**–BPMN, **2**–UML2.0 AD [21, 14], **3**–BPEL [20, 1], and **4**–Oracle BPEL PM v.10.1.2 [11]

– *Multiple resource patterns* – which describe scenarios where there is a many to many relationship between specific work items and the resources undertaking those work items.

In BPMN, the association of a particular action or set of actions with a specific resource is illustrated through the use of the Pools and Lanes constructs, commonly called Swimlanes (pp. 102-106). "A Pool represents a Participant in the Process. A Participant can be a specific business entity (e.g. a company) or can be a more general business role." (p. 103). "A Lane is a sub–partition within a Pool..." (p. 106). Hence, the direct allocation pattern (RP1) as well as the role–based allocation pattern (RP2) are directly supported. Furthermore, a partitioning of a Process into Pools and Lanes is not required, i.e., the resource allocation for the different activities is not necessarily done during design time. This means that automatic execution pattern (RP11) is also supported in BPMN.

None of the other Creation Patterns are supported within BPMN. This is a consequence of the restrictive manner in which Swimlanes are specified (i.e., only by specifying their Names, and in case of sub-division, the sub-division hierarchy) and the lack of support for relationships between distinct Swimlanes. Lack of a capability specification, integrated authorisation framework, organisational model or access to some execution history, rules out any form of support for capability–based allocation (RP8), the authorisation (RP4), organisational allocation (RP10) and history-based allocation (RP9) patterns respectively.

In a BPMN model activities become "live" once they receive the specified StartQuantity control-flow tokens. The resource associated with a given Swimlane can have multiple activities execution at the same time. There is no notion of scheduling work execution or of resources selecting the work (i.e. the activity) they wish to undertake, hence there is minimal support for the Push, Auto-start or Multiple Resource patterns within BPMN. The following set of patterns from these classes are supported:

– RP14: *Distribution by allocation - single resource* – the resource(s) associated with a given Swimlane is immediately allocated a Task/Sub-Process once it is triggered.
– RP19: *Distribution on enablement* – all activities in a Swimlane are associated with the resource responsible for the Swimlane when they are triggered.
– RP36: *Commencement on creation* – an activity is assumed to be live as soon as it receives the specified StartQuantity control-flow tokens.
– RP39: *Chained execution* – once an activity is completed, subsequent activity(ies) receive a control-flow token and are triggered immediately when the specified StartQuantity of tokens is reached.
– RP42: *Simultaneous execution* – there are no constraints on how many instances of a task specified for one Swimlane can be active at any time.

None of the Pull, Detour or Visibility patterns are supported. The results from this part of the evaluation are summarised in Table 3[6].

5 Discussion and Conclusion

There are inherent difficulties in applying the Workflow Patterns Framework for assessing a language that does not have a commonly agreed-upon formal semantics nor an execution environment. The BPMN specification [18] provides a mapping from BPMN to BPEL, for which execution engines and formalisations exist. Closer inspection however shows that the mapping to BPEL in [18] is only partial, leaving aside models with unstructured topologies as well as constructs such as OR-join and complex gateways (see [12] for a discussion). Moreover, since the mapping is described in prose, it is subject to interpretations. More generally, many ambiguities can be found in the BPMN specification due to the lack of formalisation. In our work, we documented some of these ambiguities as well as assumptions that we made to circumvent them.

The results of the Workflow Patterns analysis of BPMN are presented in tables 1, 2 and 3. A "+" indicates direct support, a "+/–" partial support, and a "–" lack of support. These tables also contain results from previous pattern-based evaluations of UML 2.0 AD [21, 14], BPEL [20, 1] and an implementation of BPEL, namely Oracle BPEL PM Version 10.1.2 [11]. It can be seen from these tables that BPMN provides direct support for the majority of the control-flow patterns and for nearly half of the data patterns, while support for the resource patterns is scant.

Along the control-flow perspective (Table 1), BPMN lacks support for the *Multiple Instances without a priori runtime knowledge* and for the *Milestone* patterns while the

[6] BPEL does not cover the resource perspective and is therefore omitted from this table

Creation Patterns	1	2	4	Pull Patterns (cont.)	1	2	4
1. Direct Allocation	+	+	+	24. System-Determ. Work Queue Content	–	–	–
2. Role-Based Allocation	+	+	+	25. Resource-Determ. Work Queue Content	–	–	+
3. Deferred Allocation	–	–	+	26. Selection Autonomy	–	–	+
4. Authorization	–	–	–	Detour Patterns			
5. Separation of Duties	–	–	–	27. Delegation	–	–	+
6. Case Handling	–	–	+	28. Escalation	–	–	+
7. Retain Familiar	–	–	+	29. Deallocation	–	–	+
8. Capability-based Allocation	–	–	+	30. Stateful Reallocation	–	–	+
9. History-based Allocation	–	–	+/–	31. Stateless Reallocation	–	–	–
10. Organizational Allocation	–	–	+/–	32. Suspension/Resumption	–	–	+
11. Automatic Execution	+	+	+	33. Skip	–	–	+
Push Patterns				34. Redo	–	–	–
12. Distribution by Offer-Single Resource	–	–	+	35. Pre-Do	–	–	–
13. Distribution by Offer-Multiple Resources	–	–	+	Auto-start Patterns			
14. Distribution by Allocation-Single Resource	+	+	+	36. Commencement on Creation	+	+	–
15. Random Allocation	–	–	+/–	37. Commencement on Allocation	–	–	–
16. Round Robin Allocation	–	–	+/–	38. Piled Execution	–	–	–
17. Shortest Queue	–	–	+/–	39. Chained Execution	+	+	–
18. Early Distribution	–	–	–	Visibility Patterns			
19. Distribution on Enablement	+	+	+	40. Config. Unallocated Work Item visibility	–	–	–
20. Late Distribution	–	–	–	41. Config. Allocated Work Item visibility	–	–	–
Pull Patterns				Multiple Resource Patterns			
21. Resource-Init. Allocation	–	–	–	42. Simultaneous Execution	+	+	+
22. Resource-Init. Exec. - Allocated Work Item	–	–	+	43. Additional Resources	–	–	+
23.Resource-Init. Execution - Offered Work Item	–	–	+				

Table 3. Support for the Resource Patterns
in **1**–BPMN, **2**–UML2.0 AD [21, 14], and **4**–Oracle BPEL PM v.10.1.2 [11]

Synchronising merge, *Discriminator* and *Interleaved parallel routing* patterns are only partially supported. The limitations in capturing the *Milestone* and *Interleaved parallel routing* patterns stem from the lack of an explicit notion of "state". States can only be captured indirectly through event-based decision Gateways. As for the *Synchronising merge*, partial support is provided by BPMN's OR-join Gateway but the semantics of this construct needs generalisation to cover unstructured process models (see [22] for a general treatment of the OR-join). Finally, the concepts available to describe discriminator and tasks and sub-processes with multiple instances require extensions to properly capture the corresponding patterns.

An outcome of the analysis of the Control-flow perspective that is not visible from Table 1 is that many patterns have multiple representations. The simpler patterns have as many as three different representations in BPMN. On the other hand, detailed knowledge of the attributes associated to BPMN's modelling constructs (which do not have a graphical representation) is required to capture some of the more advanced patterns.

Regarding BPMN's support for the Data patterns (Table 2) it can be seen that *Workflow* and *Environment data* patterns are not supported. *Data interaction to and from a Multiple Instances task* is not supported because any instance-specific data for a task or sub-process with a "multiple instance" marker can not be specified. Also support for the external data interaction patterns is limited. Only the patterns capturing the interaction between tasks and the environment are supported, as they can be captured by modelling the environment as a separate process which may be represented in full, as an abstract/public process, or implicitly through references in send and receive tasks/events.

Finally, BPMN's support for the Resource perspective is minimal (Table 3). It is acknowledged in the specification ([18], p. 22) that the modelling of organizational structures and resources is outside the scope of BPMN. However, the presence of the concepts Lane and Pool for representing parties and roles gives a contradictory impression. It is obvious though that Pools and Lanes do not provide a means for representing the subtleties associated with selective work allocation across a range of possible resources and the management of the resulting work items at run-time.

The tables also compare BPMN with UML 2.0 AD and BPEL. Along the Control-flow perspective, BPMN and UML 2.0 AD are largely overlapping. BPMN is slightly stronger when it comes to capturing the *Interleaved parallel routing* and the *Synchronising merge* patterns and slightly weaker in its support for the *Discriminator* pattern, but these differences are minor. It can also be seen from Table 1 that some Control-flow patterns are supported in BPMN but not in BPEL and vice-versa. Thus, manifestations of these patterns in a BPMN model would require special care when translating the model into BPEL. A translation from BPMN to BPEL is hence not as straightforward as it is often purported to be.

For the Data perspective the support for the patterns in BPMN and UML 2.0 AD is slightly different. UML 2.0 AD is stronger in capturing *Multiple instances data* as well as *Data interaction to and from multiple instances tasks*, while BPMN is stronger in the *Data interaction between task and environment*, due to the fact that the environment can be explicitly modelled. There are further differences for the patterns in the Data transfer and Data-based routing categories, as well as differences from the patterns captured by BPEL. However, even if the set of patterns captured in this perspective is distinct for every language and even if none of the languages fully captures all the patterns, it can be argued that the Data perspective is reasonably well covered.

Unfortunately, the same can not be said for the Resource perspective. The presence of the concepts Lane and Pool in BPMN reveals the need (and an intention) to support this perspective. However, providing support for a minimal set of resource patterns only exposes the immaturity of the language along this perspective. To the benefit of BPMN, it can be said that support for the resource perspective is also minimal in UML 2.0 AD and out of the scope of the upcoming BPEL standard. At the same time, extensions of BPEL to cover the resource perspective have been proposed (e.g. BPEL4People [9]) and some of these extensions are implemented in commercial tools such as Oracle BPEL PM, thus highlighting even further the necessity of capturing this perspective. More generally, the lack of support in BPMN and UML 2.0 AD for the resource perspective, contrasted to the ongoing efforts in the BPEL community to address this perspective, exposes a gap between contemporary process modelling tools and process execution engines (the latter generally support the resource perspective in one way or another). To achieve a consistent and coherent use of process models, from analysis down to implementation and enactment, it is important that the Resource perspective is more widely acknowledged as an integral part of business process modelling. Instead of creating new process modelling notations that largely overlap with existing ones along the control-flow perspective, the focus should rather be on further refining the existing notations to satisfactorily cover all aspects relevant to PAISs.

Acknowledgement: Thanks to Chun Ouyang for valuable discussions on BPMN during the work and to Nataliya Mulyar for her analysis of Oracle BPEL.

References

1. W.M.P. van der Aalst, M. Dumas, A. H.M. ter Hofstede, N. Russell, H. M.W Verbeek, and P. Wohed. Life After BPEL? In M. Bravetti et al., editor, *In Proc. of the 2nd Int. Workshop on Web Services and Formal Methods (WS-FM)*, volume 3670 of *LNCS*, pages 35–50. Springer Verlag, 2005.
2. W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow Patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003.
3. BPMI.org. Business Process Modeling Notation (BPMN) Information: Frequently Asked Questions. `http://www.bpmn.org/Documents/FAQ.htm`, October 2004. accessed 15 November 2005.
4. M. Dumas, W.M.P. van der Aalst, and A.H.M. ter Hofstede, editors. *Process-Aware Information Systems: Bridging People and Software Through Process Technology*. John Wiley and Sons, 2005.
5. N. Guarino. Formal Ontology and Information Systems. In *Proc. of Formal Ontology in Information Systems (FOIS'98)*, pages 3–15. IOS PRess, 1998.
6. S. Jablonski and C. Bussler. *Workflow Management: Modeling Concepts, Architecture, and Implementation*. International Thomson Computer Press, London, UK, 1996.
7. B. Kiepuszewski, A.H.M. ter Hofstede, and C. Bussler. On Structured Workflow Modelling. In B. Wangler and L. Bergman, editors, *Proc. of the 12th Int. Conference on Advanced Information Systems Engineering (CAiSE00)*, volume 1789 of *LNCS*, pages 431–445. Springer Verlag, 2000.
8. E. Kindler. On the Semantics of EPCs: A Framework for Resolving the Vicious Circle. *Data and Knowledge Engineering*, 56(1):23–40, 2006.
9. M. Kloppmann, D. Koenig, F. Leymann, G. Pfau, A. Rickayzen, C. von Riegen, P. Schmidt, and I. Trickovic. WS-BPEL Extension for People – BPEL4People. `http://www.ibm.com/developerworks/webservices/library/specification/ws-bpel4people`, July 2005. accessed 16 March 2005.
10. A. Miheev and M. Orlov. Perspectives of Workflow Systems, in Russian. `http://www.russianenterprisesolutions.com/reviews/04/112.html`, 2004. accessed 15 November 2005.
11. N.A. Mulyar. Pattern-based Evaluation of Oracle-BPEL (v.10.1.2). Technical report, Center Report BPM-05-24, BPMcenter.org, 2005.
12. C. Ouyang, M. Dumas, S. Breutel, and A.H.M. ter Hofstede. Translating Standard Process Models to BPEL. To appear in *Proceedings of 18th International Conference on Advanced Information Systems Engineering (CAiSE 2006)*, June 2006.
13. N. Russell, W.M.P. van der Aalst, A.H.M. ter Hofstede, and D. Edmond. Workflow Resource Patterns: Identification, Representation and Tool Support. In O. Pastor and J. Falcão e Cunha, editors, *Proc. of 17th Int. Conf. on Advanced Information Systems Engineering (CAiSE05)*, volume 3520 of *LNCS*, pages 216–232. Springer, 2005.

14. N. Russell, W.M.P. van der Aalst, A.H.M. ter Hofstede, and P. Wohed. On the Suitability of UML 2.0 Activity Diagrams for Business Process Modelling. In *to appear in Proc. of The 3rd Asia-Pacific Conf. on Conceptual Modelling (APCCM 2006)*. Springer Verlag, Jan 2006.
15. N. Russell, A.H.M. ter Hofstede, D. Edmond, and W.M.P. van der Aalst. Workflow Data Patterns. In L. Delcambre et al., editor, *Proc. of 24th Int. Conf. on Conceptual Modeling (ER05)*, volume 3716 of *LNCS*, pages 353–368. Springer Verlag, Oct 2005.
16. T. Wand and R. Weber. An Ontological Model of an Information System. *IEEE Transactions on Software Engineering*, 16(11):1282–1292, 1990.
17. WfMC. Workflow Management Coalition Terminology & Glossary, Document Number WFMC-TC-1011, Document Status - Issue 3.0. Technical report, Workflow Management Coalition, Brussels, Belgium, 1999.
18. S. White. Business Process Modeling Notation (BPMN). Version 1.0 - May 3, 2004, BPMI.org, 2004. `www.bpmi.org`.
19. S. White. Process Modeling Notations and Workflow Patterns. In L. Fischer, editor, *Workflow Handbook 2004*, pages 265–294. Future Strategies Inc., Lighthouse Point, FL, USA, 2004.
20. P. Wohed, W.M.P. van der Aalst, M. Dumas, and A.H.M. ter Hofstede. Analysis of Web Services Composition Languages: The Case of BPEL4WS. In Il-Y. Song et al., editor, *Proc. of 22nd Int. Conf. on Conceptual Modeling (ER 2003)*, volume 2813 of *LNCS*, pages 200–215. Springer, 2003.
21. P. Wohed, W.M.P. van der Aalst, M. Dumas, A.H.M. ter Hofstede, and N. Russell. Pattern-Based Analysis of the Control-Flow Perspective of UML Activity Diagrams. In L. Delcambre et al, editor, *Proc. of 24th Int. Conf. on Conceptual Modeling (ER05)*, volume 3716 of *LNCS*, pages 63–78. Springer Verlag, 2005.
22. M.T. Wynn, D. Edmond, W.M.P. van der Aalst, and A.H.M. ter Hofstede. Achieving a General, Formal and Decidable Approach to the OR-Join in Workflow Using Reset Nets. In G. Ciardo and P. Darondeau, editors, *Proc. of 26th Int. Conf. on Applications and Theory of Petri Nets 2005*, volume 3536 of *LNCS*, pages 423–443. Springer, 2005.