On the Choice Between Graph-Based and Block-Structured Business Process Modeling Languages

Oliver Kopp, Daniel Martin, Daniel Wutke, Frank Leymann

Institute of Architecture of Application Systems, University of Stuttgart, Germany
{kopp, martin, wutke,leymann}@iaas.uni-stuttgart.de

On the Choice Between Graph-Based and Block-Structured Business Process Modeling Languages

Oliver Kopp Daniel Martin Daniel Wutke Frank Leymann
Institute of Architecture of Application Systems
Universtät Stuttgart, Universitätsstraße 38, 70569 Stuttgart, Germany
{kopp,martin,wutke,leymann}@iaas.uni-stuttgart.de

Abstract: The most prominent business process notations in use today are BPMN, EPC and BPEL. While all those languages show similarities on the conceptual level and share similar constructs, the semantics of these constructs and even the intended use of the language itself are often quite different. As a result, users are uncertain when to use which language or construct in a particular language, especially when they have used another business process notation before. In this paper, we discuss the core characteristics of graph-based and block-structured modeling languages and compare them with respect to their join and loop semantics.

1 Introduction

Workflow technology is a central aspect of Business Process Management (BPM) and an important technology in both industry and academia. Workflows are instances of workflow models, which are representations of real-world business processes [LR00, Wes07]. Basically, a workflow model consists of activities and the ordering amongst them. Workflow models can serve different purposes: on the one hand, they can be employed for documentation of business processes itself, e.g. for facilitating business process modeling by business analysts; on the other hand, workflow models defined by IT experts can serve as input for Workflow Management Systems (WfMS) that allow their machine-aided execution. The problem of facilitating the creation of executable business process models based on abstract business process descriptions, e.g. through enhancing them with enough information to facilitate their automated execution, is known as the *Business-IT gap* [DvdAtH05]. A number of workflow languages exists for the specification and the graphical representation of processes. One important aspect is the control flow, which specifies the execution order of the activities. Conceptually, workflow languages can be classified according to whether their control flow modeling style is centered around the notion of *blocks* or the notion of *graphs*. In block-structured languages, control flow is defined similar to existing programming languages by using block-structures such as *if* or *while*. In contrast, process control flow in graph-oriented workflow languages is defined through explicit *control links* between activities.

The intended use of a workflow language places a number of requirements and restrictions on the kind of language employed; whether used primarily for documentation purposes (*abstract processes*) or whether it is used to provide a detailed process model that can be deployed on a WfMS for automatic execution (*executable processes*) highly depends on the process to be modeled and the intended use of the resulting model. Moreover, certain languages even allow for modeling abstract processes as well as executable processes. As a result, guidelines have to be provided to process modelers to allow them choosing the "right" language for their purpose.

In this paper, a number of workflow languages are compared with respect to their intended use, their notation and serialization, their basic modeling approach (block-structured vs. graph-oriented vs. hybrid), their supported structure of loops (structured loops vs. arbitrary cycles) and their support for expressing explicit data flow. Block-structured and graph-oriented workflow languages differ in their representation of loops, splits and joins. We see these aspects as the main distinction between these languages. Therefore, this paper focuses on the comparison of loops, splits and joins.

The compared workflow languages comprise *Event-driven Process Chains (EPC, [STA05, KNS92])* and the *Business Process Modeling Notation (BPMN, [Obj08])* on the side of languages targeted primarily on modeling processes for documentation purposes and the *Web Service Business Process Execution Language (BPEL, [Org07])* and the *Windows Workflow Foundation (WF, [Mic08])* as languages for modeling (also) executable processes.

1.1 Related Work

In this paper, we compare modeling languages with respect to their support of block-structured and graph-based modeling constructs. Other approaches to compare modeling languages are based on patterns. Currently, there exist control flow patterns [vdAtHKB03, Kie03], process instantiation patterns [DM08], correlation patterns [BDDW07], data handling patterns [RtHEvdA05], exception handling patterns [RvdAtH06] and service interaction patterns [BDtH05]. Workflow patterns focus on the expressiveness of the control flow constructs and do not explicitly distinguish between graph-based and block-structured modeling. The other patterns do not focus on the control flow, but on the capability of the language to specify process instantiation, process instance correlation and the handling of data, exceptions and interactions with other services, which are not captured in this work.

Different intentions of different process languages have been addressed in the context of BPMN and BPEL in [RM06] and [Pal06]. They address the intention of these modeling languages, but do not focus on the constructs to model control flow. The suitability of BPMN for business process modeling is investigated in [WvdAD$^+$06]. However, BPMN is not compared to other languages in this work.

Besides the presented languages, there are several other graph-based and block-structured languages and formalisms. A prominent formal graph-based language is the Petri-net based workflow nets [vdA98]. Pi-calculus is block-based, since it offers a parallel and a split construct. However, due to its capabilities to generate channels, it can also be used to capture the semantics of graph-based languages [PW05]. An overview of all formalisms used on the workflow area is presented in [vBK06].

Transformation strategies between block-structured and graph-based languages as well as their limitations are presented in [MLZ08]. In general, all graph-based models can be mapped to block-structured models and vice versa. Typically, a mapping from a model A to a model B and mapping the model B back results in a different model A'. The main reason is that there are different strategies for the mapping and that arbitrary cycles are not supported by block-structured languages and thus have to be "emulated" by constructs offered by the block-structured language.

1.2 Structure of the Paper

The paper is organized as follows: in Section 2 we present a business process example, which is modeled using both, graph-based and block-structured languages. In graph-based languages, there are different rules of how to join control flows during execution. In Section 3 we present an overview of the problem and the current solutions. An overview of techniques to model loops is given in Section 4. Afterwards, we present a comparison of the workflow languages in Section 5. Finally, we provide a conclusion in Section 6.

2 Exemplary Process

In this section, we present a process that shows distinct features which we will discuss in the sections to follow. The process itself serves as a running example, being re-modeled in BPEL, EPC and BPMN to exemplify the use of graph-based and block-structured modeling approaches.

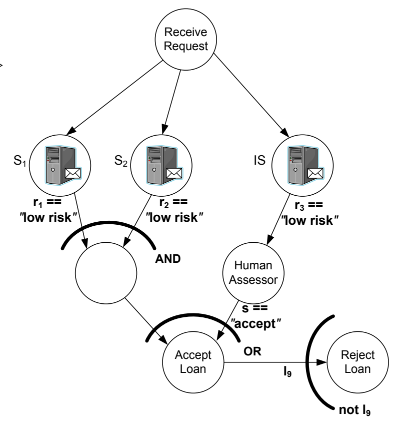
2.1 Graph-Based Modeling using BPEL `<flow>`

The process we use is a modified version of the "Loan approval" example process from [Org07], modeled using the graph-based constructs provided by BPEL. This graph-based part of BPEL originates from BPEL's predecessor WSFL [Ley01]. WSFL is based on the Flow Definition Language (FDL), formalized in [LR00] as PM-Graphs.

BPEL allows to define a workflow model using nodes and edges inside a `<flow>` element. Nodes are activities and edges are called "links". The logic of decisions and branching is solely expressed through transition conditions and join conditions. Transition conditions and join conditions are both Boolean expressions. As soon as an activity is completed, the transition conditions on their outgoing links are evaluated. The result is set as the "status of the link", which is `true` or `false`. Afterwards, the target of each link is visited. If the status of all incoming links is defined, the join condition of the activity is evaluated. If the join condition evaluates to `false`, the activity is called "dead" and the status of all its outgoing links is set to `false`. If the join condition evaluates to `true`, the activity is executed and the status of each outgoing link is evaluated. Regardless of the activity being

```
<process>
 <flow>
  <links>
   <link name="receive-to-S1">
   <link name="S1-to-AND" />
  ...
  </links>
  <receive name="ReceiveRequest">...</receive>
  <invoke ...>...</invoke>
  <empty name="AND">
   <targets>
    <joinCondition>
     $S1-to-AND AND $S2-to-AND
    </joinCondition>
    <target name="S1-to-AND" />
    <target name="S2-to-AND" />
   </targets>
   <sources>
    <source name="AND-to-OR" />
   </sources>
  </empty>
  ...
 </flow>
</process>
```

(a) BPEL code



(b) Graphical representation

Figure 1: Loan approval process in BPEL

executed, the target of each link is visited. The propagation of the dead status via `false` as link status is called dead-path elimination (DPE). DPE is conceptually detailed in [LR00], specified for BPEL in [Org07] and explained in detail in [CKLW03].

The process is presented in Figure 1: Figure 1(a) presents extracts of the BPEL code and 1(b) the graphical representation of the process. The process is initiated by the reception of a loan request at activity *receive request*. This loan request is checked in parallel by two *external* credit rating services (activities S_1 and S_2) and a company *internal* rating service IS. If the company internal rating service reports “low risk”, the subsequent activity is a risk assessment by a human assessor that manually checks the request, otherwise this step is skipped. In our case, we want to take conservative decisions, i.e. the loan request must only be accepted if either both external rating services report *low risk* or both the internal rating service and the subsequent human assessor report *low risk*. Of course we also accept the loan if all services report low risk—both external services and the internal rating service and assessor. The loan is to be rejected in any other case.

We implement these requirements using transition conditions on the links following each of the rating services which evaluate to `false` if anything else than “low risk” is reported. In case “high risk” occurs, the state of the link evaluates to `false`. The `AND`-Join following the two external rating services means that the join condition is a conjunction over the state of the links leaving from S_1 and S_2, thus the link going from the `AND`-Join to the `OR`-Join evaluates to `true` only if both external rating services returned “low risk” and therefore implements the first part of our requirements. Similarly, the result of the `OR`-Join is only `true` if one or both of its incoming links are `true`. Again, this is only the case if either both external or the internal assessment have returned “low risk”. The only part left from the requirements is to reject the loan request if it cannot be accepted. This is modeled by link l_9, which is annotated with the default transition condition `true`. The join condition on activity “reject” is a negation of the link status of link l_9 (*not* l_9). In that way, “Reject Loan” is only executed iff “Accept Loan” is not executed: l_9 is set to `true` if “Accept Loan” is executed. Thus, *not* l_9 evaluates to `false` and “Reject Loan” is not executed. If “Accept Loan” is not executed, the status of l_9 is set to `false` and *not* l_9 evaluates to `true`, leading to the execution of “Reject Loan”.

2.2 Graph-Based Modeling using BPMN and EPC

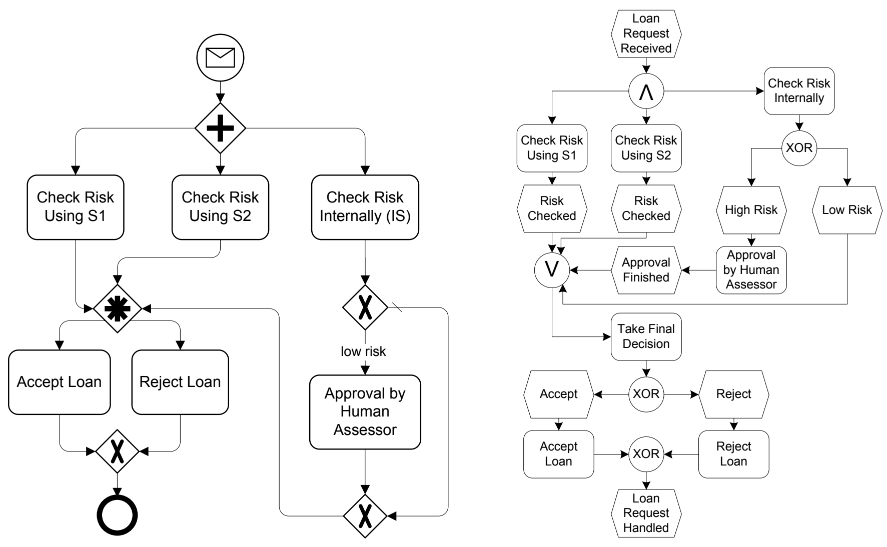
In this section, we present the “Loan approval” process introduced above, modeled using BPMN and EPC as examples of a graph-oriented modeling language. The resulting BPMN graph (Figure 2(a)) looks considerably different compared to the BPEL model presented in Section 2.1. This is mainly due to the different way of modeling joins: through arbitrary Boolean expressions in the BPEL case, or through additional explicit join constructs such as `AND`, `OR`, `XOR` in BPMN. A complex join was chosen instead of a literal translation of the process using BPMN `AND`-Joins and `OR`-Joins: the “Reject Loan” activity that has to be executed only if “Accept Loan” was not executed. This behavior cannot be modeled without being able to refer to the state of a control flow link in the join condition. In BPMN, the solution for that is to use a complex gateway that refers to variables containing the state of each of the assessment services, updated by each of the services after their completion. These variables are then used to decide which outgoing sequence flow is to follow, e.g. either reject or accept the loan request.

Since EPCs do not provide support for arbitrary join conditions, the paths of all decisions have to be merged using an `OR`-Join. Afterwards, the decision whether to accept or reject is taken at the subsequent function “Take Final Decision” (Figure 2(b). The concrete semantics has to be specified using additional text, which may be included in the diagram.

For the same reason, decisions in BPMN have to be modeled explicitly: i.e. in the BPEL graph model, we relied on dead-path elimination to skip the “human assessor” activity if the internal assessment returned “high risk”. In BPMN, this has to be modeled explicitly using a dedicated sequence flow. To summarize, the main difference between the BPEL graph-model and the BPMN model is the way how conditions are modeled: as a combination of expressions on control flow links and join conditions in the case of BPEL; or as a complex gateway in the case of BPMN. In the BPEL case, decision logic is distributed among links and activities, whereas it is represented in compact form as a complex gateway in BPMN.

2.3 Block-Structured Modeling using BPEL

Besides BPEL, the Windows Workflow Foundation (WF, [Mic08]) and “normal” programming languages support block-structured modeling. For better readability, we use a



(a) BPMN (b) EPC

Figure 2: Loan approval modeled using BPMN and EPC

simplified version of the BPEL syntax in Figure 3. We use the names of the activities as function names and abstract from their XML syntax by representing their XML attributes by function parameters. Note that our way of representing the block-structured part of BPEL emphasizes on the similarity of block structured modeling languages with regular, procedural programming languages such as C. At the expense of a different representation using programming concepts such as variables, function calls and nested block structures, this kind of modeling however provides clear semantics to every modeler familiar with basic computer programming languages. Furthermore, since the representation already is in a form similar to a “real” program, transformation into executable code typically is easier to achieve [Ecl08].

Since WS-BPEL is essentially a hybrid language that was derived from a block-structured ancestor XLANG [Tha01] and a graph-oriented ancestor WSFL [Ley01], it allows users to freely choose between both approaches. It is even possible to mix both concepts, by allowing graphs to be freely drawn within the `<flow>` element. This element may in turn may be used as a block element nested within other blocks. However, the BPEL `<flow>` can also used as a block structure only to allow for parallelism; each element it contains is executed in parallel. Using the BPEL `<flow>` as a block structure simply means not using control flow links within the block, so that each decision is represented using explicit branch or loop constructs such as `<if>` or `<while>`.

On the other hand, the way a business process typically is drawn comes very close to graph form, with nodes as activities and directed edges as control flow dependencies between them. As shown in scientific literature, it is hard to assign clear and distinct semantics to these languages (e.g. [WEvdAtH05, Men07, Weh07]) mainly due to the ambiguous way

```
sequence {
  receive(C, loan_request);
  flow {
    flow {
      extRes1 = invoke(S1, loan_request);
      extRes2 = invoke(s2, loan_request);
    }
    sequence {
      intRes = invoke(IS, loan_request);
      if (intRes=='OK') {
        intRes = invoke(assesor, loan_request);
      }
    }
  }
  if ((extRes1=='OK' && extRes2=='OK') || (intRes=='OK')) {
    invoke(CS, accept_loan);
  } else {
    invoke(CS, reject_loan);
  }
}
```

Figure 3: Loan approval modeled in block-structured BPEL

to interpret loops as well as the joins and splits they are constructed of. Sometimes a specific language even explicitly refrains from defining clear semantics (e.g. BPMN). Thus, transformation of graph-based workflow descriptions into executable form generally can be considered harder to achieve.

3 Join Conditions

As mentioned before, the way how control flow joins are implemented in a workflow modeling language heavily influences how the semantics of a certain process are expressed in the model. This section therefore revisits the examples from Section 2 and highlights different join semantics of each approach.

3.1 Kind of Join Conditions

Generally, two main types of control flow joins can be distinguished in todays workflow languages:

Restricted Choice Languages such as EPC [STA05, KNS92] and YAWL [vdAtH05] only allow to join different threads of control flow using a restricted set of operators, typically in the form of `AND`, `OR` and `XOR` elements as part of their modeling language. An important property of these languages is that it is not possible to refer to negative link state, i.e. modeling a situation as depicted in Figure 1 is not possible; a modeler has to workaround this issue, possibly creating a much more complex model. If the set of join types in a language allowing only restricted choice joins is functionally complete, a Boolean expression representing a complex join condition can be constructed using combinations of multiple join operators.

Arbitrary Expressions Languages allowing to define arbitrary Boolean expressions over the state of incoming links belong to this category. BPEL however is the only candidate that allows expressions over link state only, while BPMN allows to refer to process state (in form of process variables) in its join expressions. This has a noteworthy consequence: since it is very common to refer to process state as part of a join condition, complex join logic in BPEL has to be split among transition conditions of incoming links where process variable access is allowed, and the join condition as a Boolean expression over the state of all incoming links (and therefore the result of each of the transition conditions). In contrast to "join condition fragmentation" as in BPEL, other languages allow to model complex join conditions as one single, "compact" statement since process variable access is allowed.

BPMN is a hybrid in this case: While it offers a restricted choice (AND, OR, XOR and complex gateway), the "complex gateway" allows for defining arbitrary expressions. Naturally, restricted choice join operators are mostly used in languages whose primary intend is human-human communication of a certain process. In this case, a join refers to the availability of control flow only, in contrast to human-machine (i.e. executable) languages where joins need to be expressed in a very specific manner referring to process state and control flow and thus must be modeled in the form of a Boolean expression.

3.2 Complexity of Join Evaluation

The complexity of join evaluation has already been discussed extensively in literature. Especially, the semantics of the `OR`-Join in EPCs have raised many discussions and lead to different proposals for concrete executable semantics. An extensive presentation and comparison of the proposed semantics can be found in [Men07, MvdA07, vdAtH05, Weh07]. The inherent problem of the `OR`-Join generally is that it is hard to decide how long it should block the control flow. It is especially hard in processes containing cycles [Kin06]. Most discussions debate whether this should be resolved through local knowledge, i.e. by introducing additional arcs in the model or "negative control tokens" (as it is done by dead-path elimination) that make it possible to unblock and evaluate the join condition when it is clear that no more tokens can arrive. On the other hand, execution engines have been proposed that decide—by looking at the global state of the process—if a join can be unblocked since no more tokens will arrive on the input arcs. Naturally, these "global semantics" of join nodes introduce a significantly higher complexity of the evaluation of a join [MvdA07, DGHW07]. Languages that depend on such "global semantics" for join evaluation are BPMN and EPC.

"Local join" semantics that the execution relies on dead-path elimination (see Section 2.1) or on the introduction of additional arcs to tell the join node that no control flow will arrive on a certain path. BPEL realizes local join semantics by dead-path elimination. In that way, no additional arcs are introduced. Additional arcs are problematic when it comes to auditing the deployed process: the model deployed differs from the model finally executed by the engine.

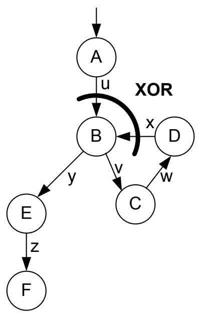


Figure 4: Example for a graph-oriented loop, modeled without an explicit loop construct through links between activities

4 Loops

A loop refers to a set of activities that are executed either while a certain loop condition holds or until a certain exit condition is reached. Two forms of loops can be found in common workflow languages: *block-structured* and *graph-based* loops. Block-structured loops, such as the *while* or *repeat until* loop, are characterized by an explicit loop construct and an exit condition at either the top or the bottom of the construct. From the process definition languages analyzed in the paper, BPEL, BPMN and WF provide support for structured loop constructs. In BPEL and WF, exit conditions can be specified to be evaluated either at the top of the loop through the *While* activity or at the bottom through the *RepeatUntil* activity. BPMN distinguishes *RepeatUntil* and *While* loops by attributes of the looping activity.

In contrast to block-structured languages, loops are modeled in graph-based languages without a dedicated loop construct by defining control flow links between activities. Typically, these links are associated with so-called *transition conditions* that define under which condition the corresponding link is to be followed by the navigator of the workflow management system. While the absence of the necessity of an explicit loop construct conceptually allows for the definition of arbitrary loops with multiple incoming and outgoing control links, such patterns are characterized by a number of problems. For instance, consider the example of a loop represented through control links between activities presented in Figure 4. In this example, link `u` denotes the loop entry and link `y` denotes the loop exit. Node `B` denotes the activity that evaluates the exit condition of the loop which repeatedly triggers execution of the loop activities `C` and `D` until the exit condition of the loop is reached and the loop is exited through link `y`. The process presented in Figure 4 can only be executed under certain assumptions. While e.g. BPEL allows for graph-based definition of process control flow within the BPEL `<flow>` construct, it does not allow for definition of graphs containing cycles; which restricts BPEL to block-structured loops. This is due to the dead-path elimination algorithm employed by BPEL [Org07]. This algorithm essentially demands each join operation—in case of the example activity `B` which joins the incoming links `u` (the loop entry) and `x` (the final link of the loop body)—to be synchronizing, i.e. to block execution of the join activity until the link status of each incoming link has been propagated to the join activity and hence the value of the join condition can be evaluated. As a result, execution of the loop can never be started, since the start of the loop depends on a defined link status of `u` which can only be produced after evaluation of activity `B`. Other approaches [MvdA07] have solved the aforementioned problem for EPCs by introducing a non-synchronizing (`XOR`) join construct and an extended form of dead/wait status propagation.

The example presented in Figure 4 also shows a second problem related to dead-path elimination in cyclic graphs in combination with arbitrary split behavior [LR00]. Node `B` not only links to node `C` (which is inside the loop) but also to the loop external node `E` (which in turn links to node `F`) through the loop exit link `y`. Assume that `B` is an `OR`-Split. In this case, after `B` is executed, it is possible that both its outgoing links are activated; as a result activities `C` and `E` are executed. Assume that execution of activity `C` takes more time than execution of `E`. After successful execution of `E`, its outgoing link `z` is activated and activity `F` is executed. Given `OR`-Split semantics in `B`, the already executed path `E`, `F` would be executed again, which might or might not be desired by the modeler. This ambiguity can be solved by restricting exit nodes on a cycle to `XOR` semantics, meaning that either the loop is exited (through one of potentially a number of exit conditions) or the loop is continued with the next cycle/iteration.

Of the analyzed languages BPMN and EPCs allow definition of arbitrary cycles.

5 Comparison

Criteria	**BPEL**	**BPMN**	**EPC**	**WF**
Intention	human-machine^{C01}	human-human	human-human	human-machine
Standardized Rendering	–	+	+	–
Standardized Serialization	+	+^{C02}	–	–^{C03}
Graph Modeling	+	+	+	–^{C04}
Well-Formed Only	–^{C05}	–	–	+
Block Modeling	+	+^{C06}	–^{C07}	n/a
Structured Loops	+	+	+	+
Arbitrary Cycles	–^{C08}	+	+	–^{C09}
Parameterized Split	+^{C10}	+	+^{C11}	n/a
Parameterized Join	+^{C12}	+^{C13}	+	n/a
Join semantics	local (DPE)	various	various	synchronization
Explicit Data Flow	–^{C14}	+	+	–

Table 1: Summarized comparison of BPEL, BPMN, EPC and WF. (revised)

In Table 1, a summary of the comparison of the workflow languages BPEL, BPMN, EPC and WF is presented with respect to their intention, standardized rendering and serialization, modeling paradigm, supported loops, splits, joins and whether they support explicit data flow. Many of the decisions are commented later in this section and referenced by Cxx, with xx standing for the number of the comment. The criteria are explained in the following. *Intention* expresses whether the respective language has been designed primarily for human-human or human-machine communication. While languages classified as human-human are used mostly for business process documentation purposes, languages classified as human-machine are used for automatic execution of business processes. As such, they require a clearly defined execution semantics that gives precise and unambiguous instructions on how a process must be executed. Note that while BPEL's abstract process profiles also facilitate its use as a modeling language, it has been classified as human-machine, since its primary focus is on executable processes (C01). *Standardized rendering* and *standardized serialization* refer to whether the language standard defines a graphical notation or a machine-processable textual representation, respectively. Note that XPDL [Wor08] is the proposed standard serialization format for BPMN diagrams (C02). The WF is a proprietary language and thus does not provide a standardized serialization; process models are directly translated to executable code (C03). Apart from WF, all compared languages support *graph-oriented modeling* of process control flow with a restriction to acyclic graphs in BPEL due to the reasons outlined in Section 4. WF is restricted to purely block-structured modeling (C04). Languages that only allow well-formed process models restrict consecutive split and join operations to the same type are referred to as *well-formed* [vdA98]. Well-formed means for example that if control flow is split using a XOR-Split it must be joined through a XOR-Join; joining a XOR-Split with an AND-Join is disallowed. In BPEL arbitrary Boolean join conditions on the status of incoming links can be specified (including in particular those resulting in non well-formed process models, C05), BPMN and EPC themselves do not define any restrictions on the types of consecutive split/join pairs. *Block-structured modeling* constructs are supported by BPEL and to a limited extent also by BPMN: BPMN supports a *while* construct and sub-processes as the only block-structured constructs (C06). EPCs offer to emulate a block construct by a pairing of connectors, but do not offer first-class block-constructs (C07). All compares languages allow for structured loops; while BPMN allows for modeling loops both as graphs and through blocks, modeling structured loops in BPEL is limited to blocks (due to the aforementioned required acyclicity of graphs in BPEL, C08). For similar reasons BPEL does not allow modeling of *arbitrary cycles* (see Section 4); as a result loops have to be modeled using blocks. In WF arbitrary cycles have to realized using state machine-based modeling (C09), which is also possible in the case of BPEL. *Parameterized split* refers to the ability to specify the link status individually for each of potentially multiple outgoing links of an activity. In BPEL this can be achieved through different transition conditions on the individual links (where an exclusive split needs mutually exclusive transition conditions, C10). EPCs are restricted to AND-Splits, OR-Splits and XOR-Splits (C11). The same restrictions hold for EPCs with respect to their support of *parameterized join* operations, i.e. the ability of defining a join condition (see Section 3). Join conditions in BPEL are restricted to Boolean expressions over the status of incoming links of the join activity (C12); BPMN allows for defining join conditions also on process instance data (C13). Note that this functionality of BPMN can be emulated in BPEL by defining appropriate transition conditions on the incoming links themselves. BPEL, as an executable process language, has a precisely defined *join semantics* while BPMN and EPC as languages focused primarily on process modeling do not. However, a number of execution semantics (including join semantics in particular) have been proposed for BPMN and EPC to fill this gap ([MvdA07, Weh07]). In order to be more generic, we use the semantics described in the respective specification of the language for comparison, not the various proposed executable interpretations or restrictions. WF only provides a block construct for parallel execution which completes its execution once each enclosed activity is completed. All compared languages express activity ordering through modeling process control flow. BPMN and EPC offer associations with data objects and thus allow to specify *explicit data flow*. In [KL06], *BPEL-D* has been proposed as an extension of BPEL that allows defining explicit data flow (C14).

6 Conclusion

In the paper we presented a comparison of four common languages for modeling business processes—BPEL, BPMN, EPC, and WF—with different fields of application and different modeling approaches. We specifically showed that BPEL supports both, block-structured and graph-based modeling. The implications of graph-based and block-structured modeling have been discussed by providing examples that highlight the languages' key characteristics. Special attention has been paid to discussing problems related to joining multiple execution paths and loops. A summary of the comparison was given in Table 1. Based on these results, we are going to provide scenarios to guide process modelers to choose a process description language fulfilling their individual requirements. We specifically focused on graph-oriented and block-structured modeling based on a simple example. However the chosen example cannot cover the full problem at the appropriate level of detail. We therefore propose a survey of the suitability of the different modeling approaches for disparate user groups (business analysts, IT architects and others).

Acknowledgments This work is supported by the BMBF funded project Tools4BPEL (01ISE08B) and the EU funded project TripCom (FP6-027324).

References

[BDDW07] A. P. Barros, G. Decker, M. Dumas, and F. Weber. Correlation Patterns in Service-Oriented Architectures. In *Proceedings of the 9th International Conference on Fundamental Approaches to Software Engineering (FASE)*, LNCS, pages 245–259, 2007.

[BDtH05] A. Barros, M. Dumas, and A. H. M. ter Hofstede. Service Interaction Patterns. In *Proceedings of the 3rd International Conference on Business Process Management*, LNCS, pages 302–318, 2005.

[CKLW03] Francisco Curbera, Rania Khalaf, Frank Leymann, and Sanjiva Weerawarana. Exception Handling in the BPEL4WS Language. In *International Conference on Business Process Management*, volume 2678 of *LNCS*, pages 276–290, 2003.

[DGHW07] Marlon Dumas, Alexander Grosskopf, Thomas Hettel, and Moe Thandar Wynn. Semantics of Standard Process Models with OR-Joins. In *Proceedings 15th Inter-*

national Conference on Coopartive Information Systems (CoopIS), volume 4803 of *LNCS*, pages 41–58, 2007.

[DM08] G. Decker and J. Mendling. Instantiation Semantics for Process Models. In *Proceedings of the 6th International Conference on Business Process Management (BPM)*, LNCS, pages 164–179, 2008.

[DvdAtH05] M. Dumas, W. M. P. van der Aalst, and A. H. M. ter Hofstede. *Process Aware Information Systems: Bridging People and Software Through Process Technology*. Wiley-Interscience, 2005.

[Ecl08] Eclipse Foundation. BPEL to Java (B2J) Subproject, 2008. `http://www.eclipse.org/stp/b2j/`.

[Kie03] B. Kiepuszewski. *Expressiveness and Suitability of Languages for Control Flow Modelling in Workflows*. PhD thesis, Queensland University of Technology, Brisbane, Australia, 2003.

[Kin06] E. Kindler. On the Semantics of EPCs: A Framework for Resolving the Vicious Circle. *Data and Knowledge Engineering*, 56(1):23–40, 2006.

[KL06] Rania Khalaf and Frank Leymann. Role-based Decomposition of Business Processes using BPEL. In *Proceedings of the IEEE International Conference on Web Services (ICWS '06)*, pages 770–780. IEEE Computer Society, 2006.

[KNS92] G. Keller, M. Nüttgens, and A.-W. Scheer. Semantische Prozeßmodellierung auf der Grundlage Ereignisgesteuerter Prozeßketten (EPK). Veröffentlichungen des Instituts für Wirtschaftsinformatik, 1992.

[Ley01] F. Leymann. Web Services Flow Language (WSFL 1.0), 2001. IBM Software Group.

[LR00] F. Leymann and D. Roller. *Production Workflow: Concepts and Techniques*. Prentice Hall PTR, 2000.

[Men07] J. Mendling. *Detection and Prediction of Errors in EPC Business Process Models*. PhD thesis, Vienna University of Economics and Business Administration, 2007.

[Mic08] Microsoft. Windows Workflow Foundation, 2008. `http://www.microsoft.com/net/WFDetails.aspx`.

[MLZ08] J. Mendling, K. B. Lassen, and U. Zdun. On the Transformation of Control Flow between Block-Oriented and Graph-Oriented Process Modeling Languages. *International Journal of Business Process Integration and Management (IJBPIM)*, 3(2), September 2008.

[MvdA07] J. Mendling and W. M. P. van der Aalst. Formalization and Verification of EPCs with OR-Joins Based on State and Context. In *Proceedings of the the 19th International Conference on Advanced Information Systems Engineering (CAiSE 2007)*, volume 4495 of *LNCS*, pages 439–453, 2007.

[Obj08] Object Management Group. *Business Process Modeling Notation, V1.1*, 2008. `http://www.omg.org/spec/BPMN/1.1/PDF`.

[Org07] Organization for the Advancement of Structured Information Standards (OASIS). *Web Services Business Process Execution Language Version 2.0 – OASIS Standard*, 2007. `http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html`.

[Pal06] N. Palmer. Understanding the BPMN-XPDL-BPEL Value Chain. *Business Integration Journal*, November/December 2006.

[PW05] F. Puhlmann and M. Weske. Using the pi-Calculus for Formalizing Workflow Patterns. In *Proceedings of the 4th International Conference on Business Process Management (BPM 2006)*, volume 4102 of *LNCS*, pages 414–419, 2005.

[RM06] J. Recker and J. Mendling. On the Translation between BPMN and BPEL: Conceptual Mismatch between Process Modeling Languages. In *CAiSE 2006 Workshop Proceedings – Eleventh International Workshop on Exploring Modeling Methods in Systems Analysis and Design (EMMSAD 2006)*, 2006.

[RtHEvdA05] N. Russell, A. H. M. ter Hofstede, D. Edmond, and W. M. P. van der Aalst. Workflow Data Patterns: Identification, Representation and Tool Support. In *24th International Conference on Conceptual Modeling (ER 2005)*, volume 3716 of *LNCS*, 2005.

[RvdAtH06] N. Russell, W. M. P. van der Aalst, and A. H. M. ter Hofstede. Workflow Exception Patterns. In *Advanced Information Systems Engineering (AISE)*, volume 4001 of *LNCS*, pages 288–302, 2006.

[STA05] A.-W. Scheer, O. Thomas, and O. Adam. *Process Aware Information Systems: Bridging People and Software Through Process Technology*, chapter Process Modeling Using Event-Driven Process Chains. Wiley-Interscience, 2005.

[Tha01] S. Thatte. *XLANG Web Services for Business Process Design*. Microsoft Corporation, 2001.

[vBK06] F. van Breugel and M. Koshkina. Models and Verification of BPEL. `http://www.cse.yorku.ca/~franck/research/drafts/tutorial.pdf`, 2006.

[vdA98] W. M. P. van der Aalst. The Application of Petri Nets to Workflow Management. *The Journal of Circuits, Systems and Computers*, 8:21–66, 1998.

[vdAtH05] W. M. P. van der Aalst and A. H. M. ter Hofstede. YAWL: Yet Another Workflow Language. *Information Systems*, 30(4):245–275, 2005.

[vdAtHKB03] 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.

[Weh07] J. Wehler. Boolean and free-choice semantics of Event-driven Process Chains. In *Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten (EPK 2007)*, pages 77–96, 2007.

[Wes07] M. Weske. *Business Process Management: Concepts, Languages, Architectures*. Springer, Berlin, 2007.

[WEvdAtH05] 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 *Applications and Theory of Petri Nets*, volume 3526 of *LNCS*, pages 423–443, 2005.

[Wor08] Workflow Management Coalition. *XML Process Definition Language Version 2.1*, 2008. `http://www.wfmc.org/xpdl-developers-center.html`.

[WvdAD+06] P. Wohed, W. M. P. van der Aalst, M. Dumas, A. H. M. ter Hofstede, and N. Russell. On the Suitability of BPMN for Business Process Modelling. In *Fourth International Conference on Business Process Management (BPM)*, volume 4102 of *LNCS*, pages 161–176, 2006.

All links were followed on 10/13/08.