Institute of Architecture of Application Systems

The Difference Between Graph-Based and Block-Structured Business Process Modelling Languages

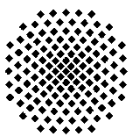
Oliver Kopp, Daniel Martin, Daniel Wutke, Frank Leymann

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

BibTeX:

```
@article{ART-2009-10,
  author = {Oliver Kopp and Daniel Martin and Daniel Wutke and Frank
Leymann},
  title  = {The Difference Between Graph-Based and Block-Structured
Business Process Modelling Languages},
  journal = {Enterprise Modelling and Information Systems},
  editor = {Ulrich Frank},
  year   = {2009},
  pages  = {3--13},
  volume = {4},
  number = {1},
  publisher = {Gesellschaft f\"{u}r Informatik e.V. (GI)}
}
```


See also MoBIS-Homepage: http://www.wi-inf.uni-duisburg-essen.de/MobisPortal



Universität Stuttgart
Germany

Oliver Kopp, Daniel Martin, Daniel Wutke, Frank Leymann

The Difference Between Graph-Based and Block-Structured Business Process Modelling Languages

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 modelling 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 [LeRo00], [Wesk07]. 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 modelling 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 [DuAa05]. 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 modelling 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 modelled and the intended use of the resulting model. Moreover, certain languages even allow for modelling abstract processes as well as executable processes. As a result, guidelines have to be provided to process modellers 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 serialisation, their basic modelling 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, [ScTh05], [KeNü+92]) and the Business Process Modelling Notation (BPMN, [OMG09]) on the side of languages targeted primarily on modelling processes for documentation purposes

and the Web Service Business Process Execution Language (BPEL, [OASIS07]) and the Windows Workflow Foundation (WF, [Micr09]) as languages for modelling (also) executable processes.

1.1 Related Work

In this paper, we compare modelling languages with respect to their support of block-structured and graph-based modelling constructs. Other approaches to compare modelling languages are based on patterns. Currently, there exist control flow patterns [AaHo+03], [Kiep03], process instantiation patterns [DeMe08], correlation patterns [BaDe+07], data handling patterns [RuHo+05], exception handling patterns [RuAa06] and service interaction patterns [BaDu05]. Workflow patterns focus on the expressiveness of the control flow constructs and do not explicitly distinguish between graph-based and block-structured modelling. 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 [ReMe06] and [Palm06]. They address mainly the intention of these modelling languages and do not focus on the constructs to model control flow. The suitability of BPMN for business process modelling is investigated in [WoAa+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 [Aals98]. 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 [PuWe05]. An overview of all formalisms used on the workflow area is presented in [BrKo06].

There is research whether visual programming or textual programming is more suited to model and understand programs. The experiments presented in [CuTa87], [ChKu01] show that "visual representations [outperform] the textual program". In the case of flow-charts the same result is presented in [Scan89] and the experience report presented in [BaHa95] shows that the productivity of visual programming outperforms textual programming. [KiAu97] shows that "graphics may be better for technical, non-programmers than they are for programmers because of the great amount of experience that programmers have with textual notations in programming languages". Finally, the studies presented in [GrPe91, GrPe92, MoMa+93] show that "graphics [is] significantly slower than text" [Petr95]. All in all, it is not finally proofed that visual programming is (in all cases) more suitable than textual programming. There are no studies specific to the languages compared in this paper and no research, whether modellers are more effective in modelling and in understanding models expressed in graphs or in block-structures.

1.2 Structure of the Paper

The paper is organised as follows: in Section 3 we present a business process example, which is modelled 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 4 we present an overview of the problem and the current solutions. An overview of techniques to model loops is given in Section 5. Subsequently, we present a comparison of the workflow languages in Section 6. Finally, we provide a conclusion in Section 8.

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-modelled in BPEL, EPC and BPMN to exemplify the use of graph-based and block-structured modelling approaches.

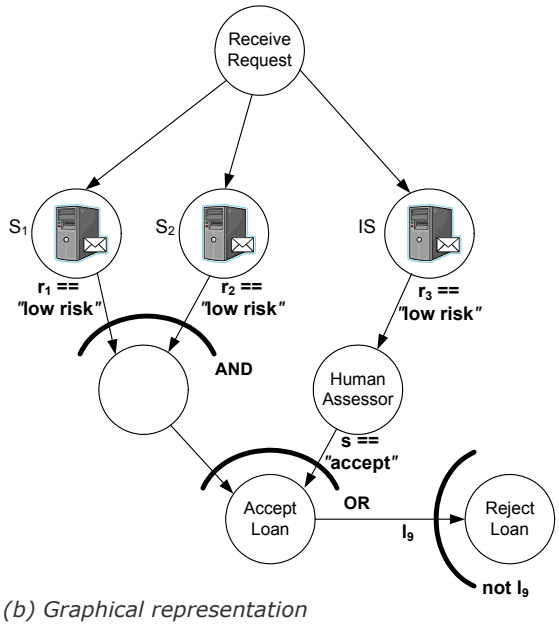
2.1 Graph-based Modelling using BPEL <flow>

The process we use is a modified version of the "Loan approval" example process from [OASIS07], modelled using the graph-based constructs provided by BPEL. This graph-based part of BPEL originates from BPEL's predecessor WSFL [Leym01]. WSFL is based on the Flow Definition Language (FDL), formalized in [LeRo00] 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

```
<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

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 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 [LeRo00], specified for BPEL in [OASIS07] and explained in detail in [CuKh+03].

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 S1 and S2) 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 S1 and S2, 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 modelled by link l9, which is annotated with the default transition condition true. The join condition on activity "reject" is a negation of the link status of link l9 (not l9). In that way, "Reject Loan" is only executed iff "Accept Loan" is not executed: l9 is set to true if "Accept Loan" is executed. Thus, not l9 evaluates to false and "Reject Loan" is not executed. If "Accept Loan" is not executed, the status of l9 is set to false and not l9 evaluates to true, leading to the execution of "Reject Loan".

2.2 Graph-Based Modelling using BPMN and EPC

In this section, we present the "Loan approval" process introduced above, modelled using BPMN and EPC as examples of a graph-oriented modelling language. The resulting BPMN graph (Figure 2) looks considerably different compared to the BPEL model presented in Section 3.1. This is mainly due to the different way

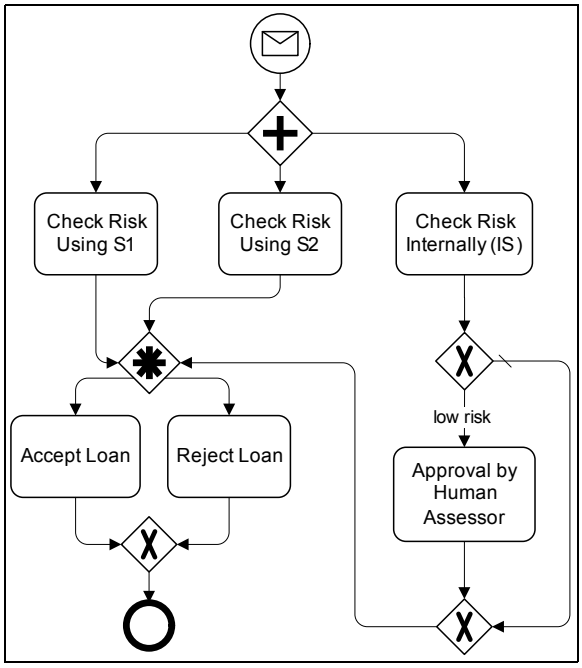


Figure 2: Loan approval modelled using BPMN

of modelling 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 behaviour cannot be modelled 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 3). 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 modelled 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 modelled explicitly using a dedicated sequence flow.

To summarise, the main difference between the BPEL graph-model and the BPMN model is the way how

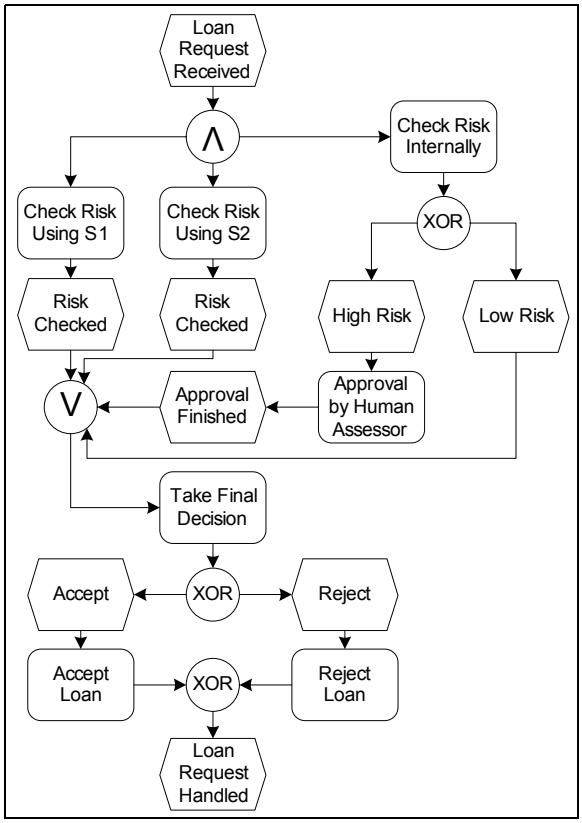


Figure 3: Loan approval modelled using EPC

conditions are modelled: 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 Modelling using BPEL

Besides BPEL, the Windows Workflow Foundation (WF, [Micr09]) and "normal" programming languages support block-structured modelling. For better readability, we use a simplified version of the BPEL syntax in Figure 4. 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 emphasises on the similarity of block structured modelling 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 modelling however provides clear semantics to every modeller familiar with basic computer programming languages. Furthermore, since the representation already is in a

```
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 4: Loan approval modelled in block-structured BPEL

form similar to a "real" program, transformation into executable code typically is easier to achieve [Ecli09].

Since WS-BPEL is essentially a hybrid language that was derived from a block-structured ancestor XLANG [That01] and a graph-oriented ancestor WSFL [Leym01], 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. [WyEd+05], [Mend07], [Wehl07]) mainly due to the ambiguous way 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 Condition

As mentioned before, the way how control flow joins are implemented in a workflow modelling language heavily influences how the semantics of a certain process are expressed in the model. This section therefore revisits the examples from Section 3 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 [ScTh05], [KeNü+92] and YAWL [AaHo05] 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 modelling language. An important property of these languages is that it is not possible to refer to negative link state, i.e. modelling a situation as depicted in Figure 1 is not possible; a modeller has to work around 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 Expression 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 modelled 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 [Mend07], [MeAa07], [AaHo05], [Wehl07]. 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 [Kind06]. 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 [MeAa07], [DuGr+07]. Languages that depend on such "global semantics" for join evaluation are BPMN and EPC.

"Local join" semantics means that the execution relies on dead-path elimination (see Section 3.1) or on the introduction of additional arcs to tell the join node that no control flow will arrive on a certain path. BPEL realises 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.

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 characterised by an explicit loop construct and an exit condition at either the top or the bottom of the construct. From the process definition languages analysed 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 <repeat until> activity. BPMN distinguishes repeat until and while loops by attributes of the looping activity.

In contrast to block-structured languages, loops are modelled 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 characterised by a number of problems. For instance, consider the example of a loop represented through control links between activities presented in Figure 5. 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.

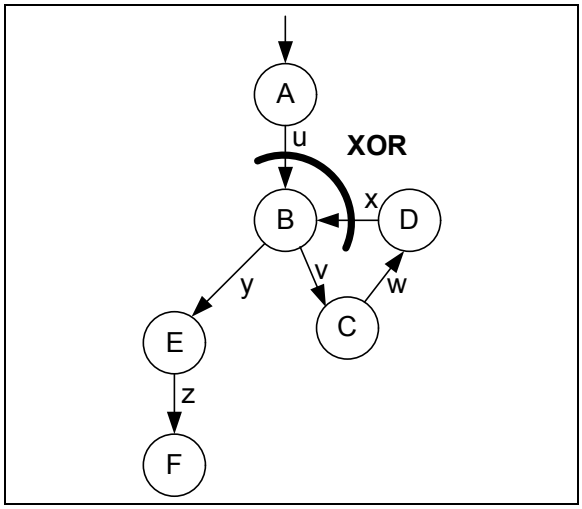


Figure 5: Example for a graph-oriented loop, modelled without an explicit loop

The process presented in Figure 5 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 [OASIS07]. 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 synchronising, 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 [MeAa07] have solved the aforementioned problem for EPCs by introducing a non-synchronising (XOR) join construct and an extended form of dead/wait status propagation.

The example presented in Figure 5 also shows a second problem related to dead-path elimination in cyclic graphs in combination with arbitrary split behaviour [LeRo00]. 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 modeller. 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.

It is important to note that unstructured loops can be mapped to structured loops, using a method proposed in [ZhHa+06]. The presented approach derives a finite automaton from the unstructured loop, applies reduction rules and then generates a set of semantically equivalent, non-reducible structured control flow statements. These statements can then be represented by block-structured statements in the respective process execution language. In the presented BPEL example for instance, combinations of <sequence>, <if>, <while> and <switch> are used. As a result, languages that only support structured loops can still support unstructured loops as well, given they support the structured control flow statements necessary to express the result of the loop transformation.

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

5 Comparison

In Table 1, a summary of the comparison of the workflow languages BPEL, BPMN, EPC and WF is presented with respect to their intention, standardised rendering and serialisation, modelling 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 modelling language, it has been classified as human-machine, since its primary focus is on executable processes (C01). Standardised rendering and standardised serialisation 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 serialisation format for BPMN diagrams (C02). The WF is a proprietary language and thus does not provide a standardised serialisation; process models are directly translated to executable code (C03). Apart from WF, all compared languages support graph-oriented modelling of process control flow with a restriction to acyclic graphs in BPEL due to the reasons outlined in Section 5. WF is restricted to purely block-structured modelling (C04). Languages that only allow well-formed process models restrict

Criteria	BPEL	BPMN	EPC	WF
Intention	human-machine^{C01}	human-human	human-human	human-machine
Standardised Rendering	-	+	+	-
Standardised Serialisation	+	+^{C02}	-	-^{C03}
Graph Modelling	+	+	+	-^{C04}
Well-Formed Only	-^{C05}	-	-	+
Block Modelling	+	+^{C06}	-^{C07}	+^{C04}
Structured Loops	+	+	+	+
Arbitrary Cycles	-^{C08}	+	+	-^{C09}
Parameterised Split	+^{C10}	+	+^{C11}	n/a
Parameterised Join	+^{C12}	+^{C13}	+	n/a
Join Semantics	local (DPE)	various	various	synchronisation
Explicit Data Flow	-^{C14}	+	+	-

Table 1: Summarised comparison of BPEL, BPMN, EPC and WF

consecutive split and join operations to the same type are referred to as well-formed [Aals98]. 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 modelling 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 modelling loops both as graphs and through blocks, modelling 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 modelling of arbitrary cycles (see Section 5); as a result loops have to be modelled using blocks. In WF arbitrary cycles have to realised using state machine-based modelling (C09), which is also possible in the case of BPEL. Parameterised 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 parameterised join operations, i.e. the ability of defining a join condition (see Section 4). 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 modelling do not. However, a number of execution semantics (including join semantics in particular) have been proposed for BPMN and EPC to fill this gap [MeAa07], [Wehl07], [DuGr+07], [BöSö+09]. 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 modelling process control flow. BPMN and EPC offer associations with data objects and thus allow to specify explicit data flow. In [KhLe06], BPEL-D has been proposed as an extension of BPEL that allows defining explicit data flow (C14).

6 Spectrum of Process Modelling Languages

Figure 6 depicts the spectrum of process modelling languages, ranging from custom languages for documentation purposes to languages that allow for automatic execution of processes in so-called Workflow Management Systems. The presented languages can be classified into three groups according to their intention: schema-less process documentation, process documentation based on a defined process model, and executable processes. The decision which process modelling language to chose for describing certain process models depends on their intended use and whether they serve primarily documentation purposes or should be used for automatic execution.

Languages for schema-less ad-hoc process documentation, as shown on the left-hand side of Figure 6, allow for rich and flexible annotation of process models with arbitrary information and thus provide greatest flexibility with regard to the chosen representation of the process model. However, this flexibility requires modellers to precisely define the semantics of their annodations. Similar to the Entity-Relationship Model for describing the structure of relational databases, pre-defined process modelling languages, such as BPMN and EPC, assist process modellers by providing them with a set of elements with a defined syntax (and/or graphical representation) and semantics. On the one hand this results in a loss of flexibility when documenting processes, while on the other hand it may (i) lead to reduced documentation effort due to the use of graphical process modelling tools, (ii) reduce the ambiguity of the defined process models due to the restriction to a set of well-defined process modelling elements, and is (iii) less error-prone due to the suitability for process model validation techniques. The languages depicted on the right-hand side of the spectrum, BPEL and WF, are languages for describing automatically executable processes and are characterised by the most restrictive set of language rules and unambiguous execution semantics.

For scenarios in which a particular process should be documented as well as automatically executed, a number of so-called model transformations between processes described using pre-defined modelling languages and languages for describing executable processes exist. Reasons for model transformation include scenarios in which a process has been initially modelled on an abstract level using a language such as BPMN or EPC and the process should be refined for automatic execution in a later stage in the business

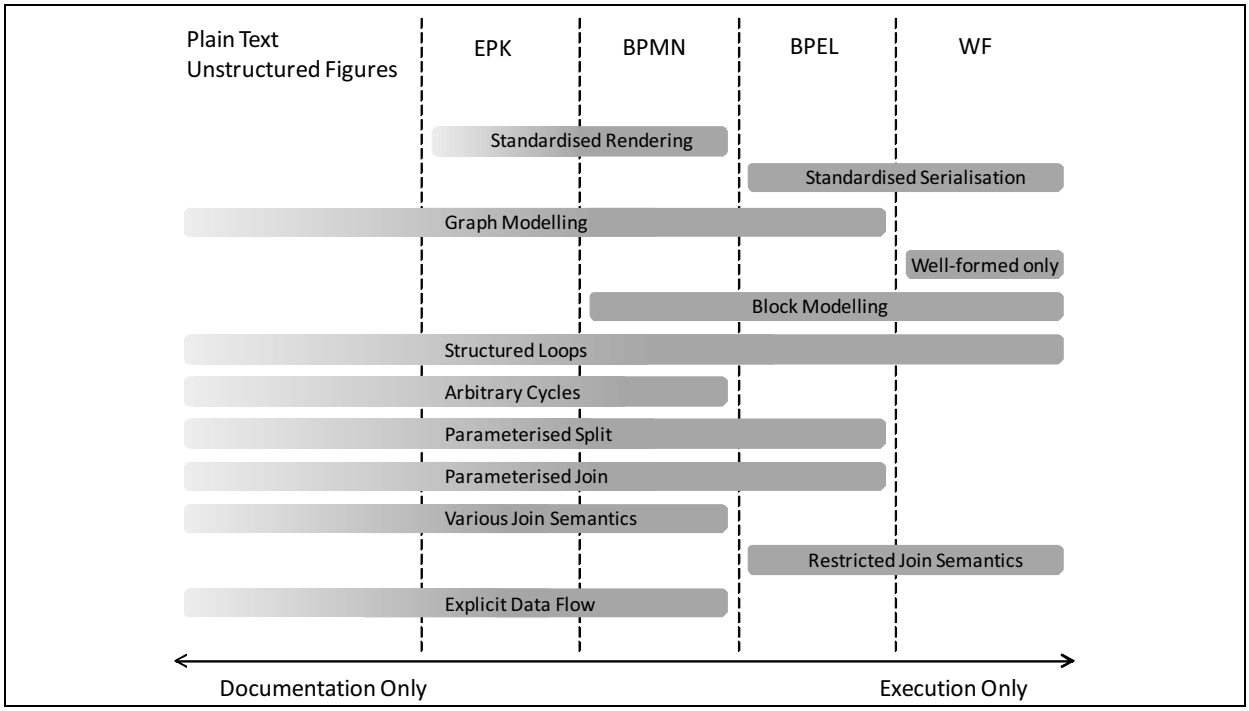


Figure 6: Spectrum of Business Process Languages and Features

process lifecycle or for visualisation purposes. An example use-case for the latter is the rendering of BPEL processes in BPMN stencils as shown in [ScKa+09], [WeDe+08]. A taxonomy for model transformations is provided in [MeGo06]. The most important criteria is the distinction between horizontal and vertical transformation. In a horizontal transformation, the model is transformed to another model on the same abstraction level. In a vertical transformation, the target model resides on a different level of abstraction. A number of approaches for transformation between different non-executable and executable process modelling languages exist [OuDu+07], [StKü+08]. Generally, such transformations are problematic since e.g. BPEL and BPMN have different target communities (technical analysts vs. business analysts) and are employed on different stages on the BPM lifecycle [ReMe06]. Another example is the mapping of EPCs to BPEL, a general overview of all available transformations and their classification using the taxonomy of [MeGo06] is given in [StKü+08].

Transformation strategies between block-structured and graph-based languages as well as their limitations are presented in [MeLZ08]. 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. Such "emulation" is sketched in Section 5 and described in detail in [ZhHa+06].

7 Conclusion

In the paper we presented a comparison of four common languages for modelling business processes—BPEL, BPMN, EPC, and WF—with different fields of application and different modelling approaches. We specifically showed that BPEL supports both, block-structured and graph-based modelling. The implications of graph-based and block-structured modelling 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 as well as identifying differences of graph-oriented and block-structured modelling languages.

A summary of the comparison was given in Table 1. Based on these results, we classified these languages according to their execution capability in a spectrum that ranges from schema-less documentation to automatic execution.

The most interesting point in the spectrum of languages is the position between BPEL and BPMN. While BPEL is a language geared towards automated execution of process models, BPMN is used mainly for process documentation but the one closest to the verge of

executability amongst all process documentation languages. Interestingly, this is also acknowledged by the BPMN committee where executability is one area of development for the upcoming BPMN 2.0 specification [OMG08].

References

[AaHo+03] van der Aalst, W. M. P.; ter Hofstede, A. H. M.; Kiepuszewski, B.: Workflow Patterns. In: Distributed and Parallel Databases 14 (2003) 1, S. 5–51.

[AaHo05] van der Aalst, W. M. P.; ter Hofstede, A. H. M.: YAWL: Yet Another Workflow Language. In: Information Systems 30 (2005) 4, S. 245–275.

[Aals98] van der Aalst, W. M. P.: The Application of Petri Nets to Workflow Management. In: The Journal of Circuits, Systems and Computers 8 (1998), S. 21–66.

[BaDe+07] Barros, A. P.; Decker, G.; Dumas, M.: Correlation Patterns in Service-Oriented Architectures. In: Proceedings of the 9th International Conference on Fundamental Approaches to Software Engineering (FASE), LNCS 2007, S. 245–259.

[BaDu05] Barros, A.; Dumas, M.; ter Hofstede, A. H. M.: Service Interaction Patterns. In: Proceedings of the 3rd International Conference on Business Process Management, LNCS 2005, S. 302–318.

[BaHa95] Baroth, E.; Hartsough, C.: Visual programming in the real world. In: Visual object-oriented programming: concepts and environments, Manning Publications Co. 1995.

[BrKo06] van Breugel, F.; Koshkina, M.: Models and Verification of BPEL. http://www.cse.yorku.ca/~franck /research/drafts/tutorial.pdf (22 April 09).

[BöSö+09] Börger, E.; Sörensen, O.; Thalheim, B.: On defining the behavior of or-joins in business process models. Journal of Universal Computer Science (2009).

[ChKu01] Chattratichart, J.; Kuljis, J.: Some Evidence for Graphical Readership, Paradigm Preference, and the Match-Mismatch Conjecture in Graphical Programs. In: Psychology of Programming Interest Group (PPIG 2001) 2001.

[CuKh+03] Curbera, F.; Khalaf, R.; Leymann, F.: Exception Handling in the BPEL4WS Language. In: International Conference on Business Process Management, Bd. 2678 von LNCS 2003, S. 276–290.

[CuTa87] Cunniff, N.; Taylor, R. P.: Graphical vs. textual representation: an empirical study of novices' program comprehension. In: Empirical Studies of Programmers: Second Workshop, Ablex Publishing Corp., Norwood, NJ, USA 1987, S. 114–131.

[DeMe08] Decker, G.; Mendling, J.: Instantiation Semantics for Process Models. In: Proceedings of the 6th International Conference on Business Process Management (BPM), LNCS 2008, S. 164–179.

[DuAa05] Dumas, M.; van der Aalst, W. M. P.; ter Hofstede, A. H. M.: Process Aware Information Systems: Bridging People and Software Through Process Technology. Wiley-Interscience 2005.

[DuGr+07] Dumas, M.; Grosskopf, A.; Hettel, T.: Semantics of Standard Process Models with OR-Joins. In: Proceedings 15th International Conference on Coopartive Information Systems (CoopIS), Bd. 4803 von LNCS 2007, S. 41–58.

[Ecli09] Eclipse Foundation: BPEL to Java (B2J) Subproject. 2009, URL: http://www.eclipse.org/stp/b2j/ (22 April 09).

[GrPe91] Green, T R G.; Petre, M.; Bellamy, R K E.: Comprehensibility of visual and textual programs: a test of superlativism against the 'match-mismatch' conjecture. In: Empirical Studies of Programmers, Fourth Workshop, Open University, Computer Assisted Learning Research Group 1991.

[GrPe92] Green, T. R. G.; Petre, M.: When visual programs are harder to read than textual programs. In: Sixth European Conference on Cognitive Ergonomics (ECCE-6) 1992.

[KeNü+92] Keller, G.; Nüttgens, M.; Scheer, A.-W.: Semantische Prozeßmodellierung auf der Grundlage Ereignisgesteuerter Prozeßketten (EPK). Veröffentlichungen des Instituts für Wirtschaftsinformatik 1992.

[KhLe06] Khalaf, R.; Leymann, F.: Role-based Decomposition of Business Processes using BPEL. In: Proceedings of the IEEE International Conference on Web Services (ICWS '06), IEEE Computer Society 2006, S. 770–780.

[KiAu97] Kiper, J. D.; Auernheimer, B.; Ames, Charles K.: Visual Depiction of Decision Statements: What is Best for Programmers and Non-Programmers? In: Empirical Softw. Eng. 2 (1997) 4, S. 361–379.

[Kiep03] Kiepuszewski, B.: Expressiveness and Suitability of Languages for Control Flow Modelling in Workflows. Dissertation, Queensland University of Technology, Brisbane, Australia 2003.

[Kind06] Kindler, E.: On the Semantics of EPCs: A Framework for Resolving the Vicious Circle. In: Data and Knowledge Engineering 56 (2006) 1, S. 23–40.

[KoMa+08] Kopp, O.; Martin, D.; Wutke, D.: On the Choice Between Graph-Based and Block-Structured Business Process Modeling Languages. In: Modellierung betrieblicher Informationssysteme (MobIS 2008). Saarbrücken, Germany, November 27 - 28, 2008., Bd. P-141 von Lecture Notes in Informatics, Gesellschaft für Informatik e.V. (GI) 2008, S. 59–72.

[LeRo00] Leymann, F.; Roller, D.: Production Workflow: Concepts and Techniques. Prentice Hall PTR 2000.

[Leym01] Leymann, F.: Web Services Flow Language (WSFL 1.0). 2001, IBM Software Group.

[MeAa07] Mendling, J.; van der Aalst, W. M. P.: Formalization and Verification of EPCs with OR-Joins Based on State and Context. In: Proceedings of the 19th International Conference on Advanced Information Systems Engineering (CAiSE 2007), Bd. 4495 von LNCS 2007, S. 439–453.

[MeGo06] Mens, T.; van Gorp, P.: A Taxonomy of Model Transformation. In: Electronic Notes in Theoretical Computer Science 152 (March 2006), S. 125–142.

[MeLZ08] Mendling, J.; Lassen, K. B.; Zdun, U.: On the Transformation of Control Flow between Block-Oriented and Graph-Oriented Process Modeling Languages. In:

International Journal of Business Process Integration and Management (IJBPIM) 3 (September 2008) 2.

[Mend07] Mendling, J.: Detection and Prediction of Errors in EPC Business Process Models. Dissertation, Vienna University of Economics and Business Administration 2007.

[Micr09] Microsoft: Windows Workflow Foundation. 2009, URL: http://www.microsoft.com/net/WFDetails.aspx (22 April 09).

[MoMa+93] Moher, T. G.; Mak, D. C.; Blumenthal, B.: Comparing the comprehensibility of textual and graphical programs: The case of Petri nets. In: Empirical Studies of Programmers: Fifth Workshop, Ablex 1993.

[OASIS07] Organization for the Advancement of Structured Information Standards (OASIS), Web Services Business Process Execution Language Version 2.0 – OASIS Standard. 2007, URL: http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html (22 April 09).

[OMG08] Object Management Group. Business Process Modeling Notation, BPMN 2.0 RFP revised submission (BMI/08-09-04), URL: http://www.omg.org/cgi-bin/doc?bmi/2008-09-04 (22 April 09).

[OMG09] Object Management Group, Business Process Modeling Notation, V1.2. 2009, URL: http://www.omg.org/spec/BPMN/1.2/PDF (22 April 09).

[OuDu+07] Ouyang, C.; Dumas, M.; ter Hofstede, A.H.M.: Pattern-based translation of BPMN process models to BPEL web services. In: International Journal of Web Services Research (JWSR) (2007).

[Palm06] Palmer, N.: Understanding the BPMN-XPDL-BPEL Value Chain. In: Business Integration Journal (November/December 2006).

[Petr95] Petre, M.: Why looking isn't always seeing: readership skills and graphical programming. In: Commun. ACM 38 (1995) 6, S. 33–44.

[PuWe05] Puhlmann, F.; Weske, M.: Using the pi-Calculus for Formalizing Workflow Patterns. In: Proceedings of the 4th International Conference on Business Process Management (BPM 2006), Bd. 4102 von LNCS 2005, S. 414–419.

[ReMe06] Recker, J.; Mendling, J.: 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.

[RuAa06] Russell, N.; van der Aalst, W. M. P.; ter Hofstede, A. H. M.: Workflow Exception Patterns. In: Advanced Information Systems Engineering (AISE), Bd. 4001 von LNCS 2006, S. 288–302.

[RuHo+05] Russell, N.; ter Hofstede, A. H. M.; Edmond, D.: Workflow Data Patterns: Identification, Representation and Tool Support. In: 24th International Conference on Conceptual Modeling (ER 2005), Bd. 3716 von LNCS 2005.

[Scan89] Scanlan, D. A.: Structured Flowcharts Outperform Pseudocode: An Experimental Comparison. In: IEEE Software 6 (Sep 1989) 5, S. 28–36.

[ScKa+09] Schumm, D.; Karastoyanova, D.; Leymann, F.: On Visualizing and Modelling BPEL with BPMN. In: Proceedings of the 4th International Workshop on Workflow Management (ICWM2009), IEEE Computer Society 2009.

[ScTh05] Scheer, A.-W.; Thomas, O.; Adam, O.: Process Aware Information Systems: Bridging People and Software Through Process Technology. Wiley-Interscience 2005 .

[StKü+08] Stein, S.; Kühne, S.; Ivanov, K.: Business to IT Transformations Revisited. In: MDE4BPM 2008.

[That01] Thatte, S.: XLANG Web Services for Business Process Design. Microsoft Corporation, 2001.

[WeDe+08] Weidlich, M.; Decker, G.; Großkopf, A.: BPEL to BPMN: The Myth of a Straight-Forward Mapping. In: Proceedings of the 16th International Conference on Cooperative Information Systems (CoopIS 2008), LNCS 2008.

[Wehl07] Wehler, J.: Boolean and free-choice semantics of Event-driven Process Chains. In: Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten (EPK 2007) 2007, S. 77–96.

[Wesk07] Weske, M.: Business Process Management: Concepts, Languages, Architectures. Springer, Berlin 2007.

[WoAa+06] Wohed, P.; van der Aalst, W. M. P.; Dumas, M.: On the Suitability of BPMN for Business Process Modelling. In: Fourth International Conference on Business Process Management (BPM), Bd. 4102 von LNCS 2006, S. 161–176.

[Wor08] Workflow Management Coalition, XML Process Definition Language Version 2.1. 2008, URL: http://www.wfmc.org/xpdl-developers-center.html (22 April 09).

[WyEd+05] Wynn, M. T.; Edmond, D.; van der Aalst, W. M. P.: Achieving a General, Formal and Decidable Approach to the OR-Join in Workflow Using Reset Nets. In: Applications and Theory of Petri Nets, Bd. 3526 von LNCS 2005, S. 423–443.

[ZhHa+06] Zhao, W.; Hauser, R.; Bhattacharya, K.; Bryant, R. B.; Cao, F.: Compiling business processes: untangling unstructured loops in irreducible flow graphs. In: International Journal of Web and Grid Services 2 (Feb. 2006), S. 68–91.

Oliver Kopp, Daniel Martin,
Daniel Wutke, Frank Leymann

Institute of Architecture of Application Systems
University of Stuttgart
Universitätsstrasse 38
70569 Stuttgart
Germany