

# Patterns and XPDL: A Critical Evaluation of the XML Process Definition Language

Wil M.P. van der Aalst

Department of Technology Management  
Eindhoven University of Technology, The Netherlands  
w.m.p.v.d.aalst@tm.tue.nl

**Abstract.** XML Process Definition Language (XPDL) is the language proposed by the Workflow Management Coalition (WfMC) to interchange process definitions between different workflow products. The goal of XPDL is to provide a Lingua Franca for the workflow domain allowing for the import and export process definitions between a variety of tools ranging from workflow management systems to modeling and simulation tools. Starting point of XPDL is a minimal set of constructs present in most workflow products. Unfortunately, this minimal set does not offer direct support to many of the workflow patterns encountered in practice and present in more mature workflow products. To address this problem, XPDL offers vendor specific extensions. However, this approach definitely does not result in a Lingua Franca. Moreover, to date, even the semantics of the core constructs of XPDL remain undefined. This paper will analyze XPDL using a set of 20 basic workflow patterns and expose some of the semantic problems.

**Keywords:** Workflow management, Workflow management systems, Workflow patterns, XML Process Definition Language (XPDL).

## 1 Introduction

The Workflow Management Coalition (WfMC) was founded in August 1993 as a international non-profit organization. Today there are about 300 members ranging from workflow vendors and users to analysts and university/research groups. The mission of the WfMC is to promote and develop the use of workflow through the establishment of standards for workflow terminology, interoperability and connectivity between workflow products. The WfMC's reference model identifies five interfaces. One of the main activities since 1993 has been the development of standards for these interfaces. Interface 1 is the link between the so-called "Process Definition Tools" and the "Enactment Service". The Process Definition Tools are used to design workflows while the Enactment Service can execute workflows. The primary goal of Interface 1 is the import and export

of process definitions. The WfMC defines a process definition as “The representation of a business process in a form which supports automated manipulation, such as modeling, or enactment by a workflow management system. The process definition consists of a network of activities and their relationships, criteria to indicate the start and termination of the process, and information about the individual activities, such as participants, associated IT applications and data, etc.” [26]. Clearly, there is a need for process definition interchange. First of all, within the context of a single workflow management system there has to be a connection between the design tool and the execution/run-time environment. Second, there may be the desire to use another design tool, e.g., a modeling tool like ARIS or Protos. Third, for analysis purposes it may be desirable to link the design tool to analysis software such as simulation and verification tools. Fourth, the use of repositories with workflow processes requires a standardized language. Fifth, there may be the need to transfer a definition interchange from one engine to another.

To support the interchange of workflow process definitions, there has to be a standardized language [4, 11, 14, 17, 19, 20]. The WfMC started working on such a language soon after it was founded. This resulted in the Workflow Process Definition Language (WPDL) [27] presented in 1999. Although many vendors claimed to be WfMC compliant, few made a serious effort to support this language. At the same time, XML emerged as a standard for data interchange. Since WPDL was not XML-based, the WfMC started working a new language named XML Process Definition Language (XPDL). The starting point for XPDL was WPDL. However, XPDL should not be considered the XML version of WPDL. Several concepts have been added/changed and the WfMC remains fuzzy about the exact relationship between XPDL and WPDL. In October 2002, the WfMC released a “Final Draft” of XPDL [28].

In [28], the authors state “More complex transitions, which cannot be expressed using the simple elementary transition and the split and join functions associated with the from- and to- activities, are formed using dummy activities, which can be specified as intermediate steps between real activities allowing additional combinations of split and/or join operations. **Using the basic transition entity plus dummy activities, routing structures of arbitrary complexity can be specified.** Since several different approaches to transition control exist within the industry, several conformance classes are specified within XPDL. These are described later in the document.” The sentence “Using the basic transition

entity plus dummy activities, routing structures of arbitrary complexity can be specified.” triggered us to look into the expressive power of XPDL.

For a critical evaluation of XPDL, we use the set of workflow patterns described in [5, 6, 30]. We have collected a set of about 30 workflow patterns and have used 20 of these patterns to compare the functionality of 15 workflow management systems (COSA, Visual Workflow, Forté Conductor, Lotus Domino Workflow, Meteor, Mobile, MQSeries/Workflow, Staffware, Verve Workflow, I-Flow, InConcert, Changengine, SAP R/3 Workflow, Eastman, and FLOWer). The result of this evaluation reveals that (1) the expressive power of contemporary systems leaves much to be desired and (2) the systems support different patterns. Note that we do not use the term “expressiveness” in the traditional or formal sense. If one abstracts from capacity constraints, any workflow language is Turing complete. Therefore, it makes no sense to compare these languages using formal notions of expressiveness. Instead we use a more intuitive notion of expressiveness which takes the modeling effort into account. This more intuitive notion is often referred to as suitability. See [15, 16] for a discussion on the distinction between formal expressiveness and suitability.

The observation that the expressive power of the available workflow management systems leaves much to be desired, triggered the question: *How about XPDL as a workflow language?* Thus far a rigorous analysis has been missing. Note that reports such as [22] do not use objective measures to compare XPDL to other standards and languages.

The remainder of the paper is structured as follows. Section 2 provides an overview of the XPDL language. In Section 3 the language is analyzed using a basic set of 20 workflow patterns. Section 4 discusses one of the core semantical problems: The join construct. Finally, Section 5 concludes the paper after comparing XPDL with workflow management systems and other standards such as UML Activity Diagrams, BPEL4WS, BPML, WSFL, XLANG, and WSCI.

## 2 XPDL: XML Process Definition Language

XPDL [28] uses an XML-based syntax, specified by an XML schema. The main elements of the language are: `Package`, `Application`, `WorkflowProcess`, `Activity`, `Transition`, `Participant`, `DataField`, and `DataType`. The `Package` element is the container holding the other elements. The `Application` element is used to specify the applications/tools invoked by the workflow processes defined in a package. The element `WorkflowProcess` is used to define workflow processes or parts of workflow processes. A

`WorkflowProcess` is composed of elements of type `Activity` and `Transition`. The `Activity` element is the basic building block of a workflow process definition. Elements of type `Activity` are connected through elements of type `Transition`. There are three types of activities: `Route`, `Implementation`, and `BlockActivity`. Activities of type `Route` are dummy activities just used for routing purposes. Activities of type `BlockActivity` are used to execute sets of smaller activities. Element `ActivitySet` refers to a self contained set of activities and transitions. A `BlockActivity` executes such an `ActivitySet`. Activities of type `Implementation` are steps in the process which are implemented by manual procedures (`No`), implemented by one of more applications (`Tool`), or implemented by another workflow process (`Subflow`). The `Participant` element is used to specify the participants in the workflow, i.e., the entities that can execute work. There are 6 types of participants: `ResourceSet`, `Resource`, `Role`, `OrganizationalUnit`, `Human`, and `System`. Elements of type `DataField` and `DataType` are used to specify workflow relevant data. Data is used to make decisions or to refer to data outside of the workflow, and is passed between activities and subflows.

In this paper, we focus on the control-flow perspective. Therefore, we will not consider functionality related to the `Package`, `Application`, and `Participant` elements. Moreover, we will only consider workflow relevant data from the perspective of routing. Appendix A shows selected parts of the XPDL Schema [28] relevant for this paper. The listing shows the elements `Activity`, `TransitionRestriction`, `TransitionRestrictions`, `Join`, `Split`, `Transition` and `Condition`. An activity may have one of more “transition restrictions” to specify the split/join behavior. If there is a transition restriction of type `Join`, the restriction is either set to `AND` or to `XOR`. The WfMC defines the semantics of such a restriction as follows: “`AND`: Join of (all) concurrent threads within the process instance with incoming transitions to the activity: Synchronization is required. The number of threads to be synchronized might be dependent on the result of the conditions of previous `AND` split(s).” and “`XOR`: Join for alternative threads: No synchronization is required.” [28]. Similarly, there are transition restrictions of type `Split` that are set to either `AND` or `XOR` with the following semantics: “`AND`: Defines a number of possible concurrent threads represented by the outgoing `Transitions` of this `Activity`. If the `Transitions` have conditions the actual number of executed parallel threads is dependent on the conditions associated with each transition, which are evaluated concurrently.” and “`XOR`: List of Identifiers of outgoing `Transitions` of this `Activity`, representing. Alternatively executed transitions. The decision as to which single transition route is selected is dependent on the condi-

tions of each individual transition as they are evaluated in the sequence specified in the list. If an unconditional Transition is evaluated or transition with condition `OTHERWISE` this ends the list evaluation.” [28]. Appendix A also shows the definition of element `Transition`. A transition connects two activities as indicated by the `From` and `To` field and may contain a `Condition` element.

The WfMC acknowledges the fact that workflow languages use different styles and paradigms. To accommodate this, XPDL allows for vendor specific extensions of the language. In addition, XPDL distinguishes three conformance classes: non-blocked, loop-blocked, and full-blocked. These conformance classes refer to the network structure of a process definition, i.e., the graph of activities (nodes) and transitions (arcs). For conformance class non-blocked there are no restrictions. For conformance class loop-blocked the network structure has to be acyclic and for conformance class full-blocked there has to be a one-to-one correspondence between splits and joins of the same type. These conformance classes correspond to different styles of modeling. Graph based workflow languages like COSA and Staffware correspond to conformance class non-blocked. Languages such as MQSeries, WSFL, and BPEL4WS correspond to conformance class loop-blocked and block-structured languages such as XLANG are full-blocked.

A detailed introduction to XPDL is beyond the scope of this paper. For more details we refer to [28].

### 3 The Workflow Patterns in XPDL

In this section, we consider the 20 workflow patterns presented in [6], and we discuss how and to what extent these patterns can be captured in XPDL. In particular, we indicate whether the pattern is directly supported by a XPDL construct. If this is not the case, we sketch a workaround solution. Most of the solutions are presented in a simplified XPDL notation which is intended to capture the key ideas of the solutions while avoiding coding details. In other words, the fragments of XPDL definitions provided here are not “ready to be run”.

**WP1 Sequence** An activity in a workflow process is enabled after the completion of another activity in the same process. **Example:** After the activity *order registration* the activity *customer notification* is executed.

**Solution, WP1** This pattern is directly supported by the XPDL as illustrated in Listing 1. Within the process *Sequence* two activities *A* and *B* are linked through transition *AB*.

**Listing 1 (Sequence)**

```

1 <WorkflowProcess Id="Sequence">
2   <ProcessHeader DurationUnit="Y"/>
3   <Activities>
4     <Activity Id="A">
5       ...
6     </Activity>
7     <Activity Id="B">
8       ...
9     </Activity>
10  </Activities>
11  <Transitions>
12    <Transition Id="AB" From="A" To="B"/>
13  </Transitions>
14 </WorkflowProcess>

```

**WP2 Parallel Split** A point in the process where a single thread of control splits into multiple threads of control which can be executed in parallel, thus allowing activities to be executed simultaneously or in any order [17, 11]. **Example:** After activity *new cell phone subscription order* the activity *insert new subscription* in Home Location Registry application and *insert new subscription* in Mobile answer application are executed in parallel.

**WP3 Synchronization** A point in the process where multiple parallel branches converge into one single thread of control, thus synchronizing multiple threads [17]. It is an assumption of this pattern that after an incoming branch has been completed, it cannot be completed again while the merge is still waiting for other branches to be completed. Also, it is assumed that the threads to be synchronized belong to the same global process instance (i.e., to the same “case” in workflow terminology). **Example:** Activity *archive* is executed after the completion of both activity *send tickets* and activity *receive payment*. Obviously, the synchronization occurs within a single global process instance: the *send tickets* and *receive payment* must relate to the same client request.

**Solutions, WP2 & WP3** This pattern directly supported by the XPDL. This is illustrated by the example shown in Listing 2. Within the process

**Listing 2 (Parallel Split/Synchronization)**

```

1 <WorkflowProcess Id="Parallel">
2   <ProcessHeader DurationUnit="Y"/>
3   <Activities>
4     <Activity Id="A">
5       ...
6       <TransitionRestrictions>
7         <TransitionRestriction>
8           <Split Type="AND">
9             <TransitionRefs>
10              <TransitionRef Id="B"/>
11              <TransitionRef Id="C"/>
12            </TransitionRefs>
13          </Split>
14        </TransitionRestriction>
15      </TransitionRestrictions>
16    </Activity>
17    <Activity Id="B">
18      ...
19    </Activity>
20    <Activity Id="C">
21      ....
22    </Activity>
23    <Activity Id="D">
24      ...
25      <TransitionRestrictions>
26        <TransitionRestriction>
27          <Join Type="AND"/>
28        </TransitionRestriction>
29      </TransitionRestrictions>
30    </Activity>
31  </Activities>
32  <Transitions>
33    <Transition Id="AB" From="A" To="B"/>
34    <Transition Id="AC" From="A" To="C"/>
35    <Transition Id="BD" From="B" To="D"/>
36    <Transition Id="CD" From="C" To="D"/>
37  </Transitions>
38 </WorkflowProcess>

```

Parallel four activities are linked through four transitions. Transitions AB and AC link the initial activity A to the two parallel activities B and C. Note that the split in activity A is of type AND and no transition conditions are specified. Transitions BD and CD link the two parallel activities B and C to the final activity D. Note that the join in activity D is of type AND and again no transition conditions are specified.

**WP4 Exclusive Choice** A point in the process where, based on a decision or workflow control data, one of several branches is chosen. **Example:** The manager is informed if an order exceeds \$600, otherwise not.

**WP5 Simple Merge** A point in the workflow process where two or more alternative branches come together without synchronization. It is an assumption of this pattern that none of the alternative branches is ever executed in parallel with another one (if it is not the case, then see the patterns Multi Merge and Discriminator). **Example:** After the payment is received or the credit is granted the car is delivered to the customer.

**Solutions, WP4 & WP5** XPDL can address the Exclusive choice pattern (WP4) in two ways. In both cases, an activity has a split and multiple outgoing transitions. One way is to use a split of type XOR, i.e., the first transition which has no condition or a condition which evaluates to true is taken. Another way is to use split of type AND and define mutual exclusive transition conditions. Listing 3 shows a solution using the first alternative. Listing 4 shows a solution using the second alternative. In the second solution transitions AB and AC have a condition. In the first solution transitions AB and AC do not have a condition which effectively implies that always the first one (AB) is taken. Besides normal conditions based on workflow relevant data, it is also possible to use conditions of type OTHERWISE (for the default branch to be taken if all other conditions evaluate to false) and of type EXCEPTION (for specifying the branch to be taken after an exception was raised). Listings 3 and 4 also show the direct support for the Simple merge (WP5).

**WP6 Multi-Choice** A point in the process, where, based on a decision or control data, a number of branches are chosen and executed as parallel threads. **Example:** After executing the activity *evaluate damage* the activity *contact fire department* or the activity *contact insurance company* is executed. At least one of these activities is executed. However, it is also possible that both need to be executed.

**Listing 3 (Exclusive Choice/Simple Merge)**

```
1 <WorkflowProcess Id="Choice1">
2   <ProcessHeader DurationUnit="Y"/>
3   <Activities>
4     <Activity Id="A">
5       ...
6       <TransitionRestrictions>
7         <TransitionRestriction>
8           <Split Type="XOR">
9             <TransitionRefs>
10              <TransitionRef Id="AB"/>
11              <TransitionRef Id="AC"/>
12            </TransitionRefs>
13          </Split>
14        </TransitionRestriction>
15      </TransitionRestrictions>
16    </Activity>
17    <Activity Id="B">
18      ...
19    </Activity>
20    <Activity Id="C">
21      ...
22    </Activity>
23    <Activity Id="D">
24      ...
25      <TransitionRestrictions>
26        <TransitionRestriction>
27          <Join Type="XOR"/>
28        </TransitionRestriction>
29      </TransitionRestrictions>
30    </Activity>
31  </Activities>
32  <Transitions>
33    <Transition Id="AB" From="A" To="B"/>
34    <Transition Id="AC" From="A" To="C"/>
35    <Transition Id="BD" From="B" To="D"/>
36    <Transition Id="CD" From="C" To="D"/>
37  </Transitions>
38 </WorkflowProcess>
```

## Listing 4 (Exclusive Choice/Simple Merge)

```

1 <WorkflowProcess Id="Choice2">
2   <ProcessHeader DurationUnit="Y"/>
3   <Activities>
4     <Activity Id="A">
5       ...
6       <TransitionRestrictions>
7         <TransitionRestriction>
8           <Split Type="AND">
9             <TransitionRefs>
10              <TransitionRef Id="AB"/>
11              <TransitionRef Id="AC"/>
12            </TransitionRefs>
13          </Split>
14        </TransitionRestriction>
15      </TransitionRestrictions>
16    </Activity>
17    <Activity Id="B">
18      ...
19    </Activity>
20    <Activity Id="C">
21      ...
22    </Activity>
23    <Activity Id="D">
24      ...
25      <TransitionRestrictions>
26        <TransitionRestriction>
27          <Join Type="XOR"/>
28        </TransitionRestriction>
29      </TransitionRestrictions>
30    </Activity>
31  </Activities>
32  <Transitions>
33    <Transition Id="AB" From="A" To="B">
34      <Condition Type="CONDITION">
35        choice == "B" </Condition>
36    </Transition>
37    <Transition Id="AC" From="A" To="C">
38      <Condition Type="CONDITION">
39        choice == "C" </Condition>
40    </Transition>
41    <Transition Id="BD" From="B" To="D"/>
42    <Transition Id="CD" From="C" To="D"/>
43  </Transitions>
44 </WorkflowProcess>

```

## Listing 5 (Multi Choice/Synchronizing merge)

```

1 <WorkflowProcess Id="Multi-choice">
2   <ProcessHeader DurationUnit="Y"/>
3   <Activities>
4     <Activity Id="A">
5       ...
6       <TransitionRestrictions>
7         <TransitionRestriction>
8           <Split Type="AND">
9             <TransitionRefs>
10              <TransitionRef Id="AB"/>
11              <TransitionRef Id="AC"/>
12            </TransitionRefs>
13          </Split>
14        </TransitionRestriction>
15      </TransitionRestrictions>
16    </Activity>
17    <Activity Id="B">
18      ...
19    </Activity>
20    <Activity Id="C">
21      ...
22    </Activity>
23    <Activity Id="D">
24      ...
25      <TransitionRestrictions>
26        <TransitionRestriction>
27          <Join Type="AND"/>
28        </TransitionRestriction>
29      </TransitionRestrictions>
30    </Activity>
31  </Activities>
32  <Transitions>
33    <Transition Id="AB" From="A" To="B">
34      <Condition Type="CONDITION">
35        amount > 5 </Condition>
36    </Transition>
37    <Transition Id="AC" From="A" To="C">
38      <Condition Type="CONDITION">
39        amount < 10 </Condition>
40    </Transition>
41    <Transition Id="BD" From="B" To="D"/>
42    <Transition Id="CD" From="C" To="D"/>
43  </Transitions>
44 </WorkflowProcess>

```

**Solution, WP6** XPDL provides direct support for the Multi-Choice pattern as shown in Listing 5. Depending on the value of `amount` activity B and/or C is/are executed, e.g., if the value of `amount` is 8 both activities are executed, otherwise just B (`amount > 5`) or C (`amount < 10`).

**WP7 Synchronizing Merge** A point in the process where multiple paths converge into one single thread. Some of these paths are “active” (i.e. they are being executed) and some are not. If only one path is active, the activity after the merge is triggered as soon as this path completes. If more than one path is active, synchronization of all active paths needs to take place before the next activity is triggered. It is an assumption of this pattern that a branch that has already been activated, cannot be activated again while the merge is still waiting for other branches to complete. **Example:** After either or both of the activities *contact fire department* and *contact insurance company* have been completed (depending on whether they were executed at all), the activity *submit report* needs to be performed (exactly once).

**Solutions, WP7** According to [28] XPDL provides direct support for the Synchronizing merge pattern. Recall the definition of the AND restriction: “AND: Join of (all) concurrent threads within the process instance with incoming transitions to the activity: Synchronization is required. The number of threads to be synchronized might be dependent on the result of the conditions of previous AND split(s).” [28] which suggests direct support for the Synchronizing merge pattern. If this is indeed the case, then Listing 5 indeed shows an example where activity D either merges or synchronizes the two ingoing transitions depending on the number of threads activated by activity A. Unfortunately, few workflow systems that claim to support XPDL have indeed this behavior. Moreover, XPDL allows for multiple interpretations as discussed in Section 4.

**WP8 Multi-Merge** A point in a process where two or more branches reconverge without synchronization. If more than one branch gets activated, possibly concurrently, the activity following the merge is started for every action of every incoming branch. **Example:** Sometimes two or more branches share the same ending. Two activities *audit application* and *process applications* are running in parallel which should both be followed by an activity *close case*, which should be executed twice if the activities *audit application* and *process applications* are both executed.

**Solution, WP8** XPDL only allows for two types of joins: AND and XOR. The semantics of these two joins is not completely clear. A join

**Listing 6 (Multi-merge?)**

```

1 <WorkflowProcess Id="Parallel">
2   <ProcessHeader DurationUnit="Y"/>
3   <Activities>
4     <Activity Id="A">
5       ...
6       <TransitionRestrictions>
7         <TransitionRestriction>
8           <Split Type="AND">
9             <TransitionRefs>
10              <TransitionRef Id="B"/>
11              <TransitionRef Id="C"/>
12            </TransitionRefs>
13          </Split>
14        </TransitionRestriction>
15      </TransitionRestrictions>
16    </Activity>
17    <Activity Id="B">
18      ...
19    </Activity>
20    <Activity Id="C">
21      ....
22    </Activity>
23    <Activity Id="D">
24      ...
25      <TransitionRestrictions>
26        <TransitionRestriction>
27          <Join Type="XOR"/>
28        </TransitionRestriction>
29      </TransitionRestrictions>
30    </Activity>
31  </Activities>
32  <Transitions>
33    <Transition Id="AB" From="A" To="B"/>
34    <Transition Id="AC" From="A" To="C"/>
35    <Transition Id="BD" From="B" To="D"/>
36    <Transition Id="CD" From="C" To="D"/>
37  </Transitions>
38 </WorkflowProcess>

```

of type XOR will offer the Simple merge pattern. Recall that the simple merge assumes that precisely one of the incoming transitions will occur. However, XPDL allows for situations where the more incoming transitions will or may occur. Consider Listing 6. Both B and C are executed. Since activity D has a join of type XOR it can already occur when one of these two have been executed. However, it is not clear how many times activity D will occur (and when). In [28] it is stated that “The XOR join initiates the Activity when the transition conditions of any (one) of the incoming transitions evaluates true.”. Since it is not specified what should happen if multiple incoming transitions evaluate to true at the same time, we conclude that XPDL does not support the Multi-Merge (WP8). See [6] for typical work-arounds.

**WP9 Discriminator** A point in the workflow process that waits for one of the incoming branches to complete before activating the subsequent activity. From that moment on it waits for all remaining branches to complete and “ignores” them. Once all incoming branches have been triggered, it resets itself so that it can be triggered again (which is important otherwise it could not really be used in the context of a loop). **Example:** To improve query response time a complex search is sent to two different databases over the Internet. The first one that comes up with the result should proceed the flow. The second result is ignored.

**Solution, WP9** XPDL allows for situations where multiple incoming transitions will or may occur. However, the precise semantics of a join of type XOR is not specified and, similar to WP8, we conclude that the Discriminator (WP9) is not supported.

**WP10 Arbitrary Cycles** A point where a portion of the process (including one or more activities and connectors) needs to be “visited” repeatedly without imposing restrictions on the number, location, and nesting of these points. Note that block-oriented languages and languages providing constructs such as “while do”, “repeat until” typically impose such restrictions, e.g., it is not possible to jump from one loop into another loop.

**Solution, WP10** XPDL distinguishes three conformance classes: non-blocked, loop-blocked, and full-blocked. Conformance class “non-blocked” directly supports this pattern. Note that the transitions basically define a relation and allow for any graph including cyclic ones. For the other conformance classes this is not allowed. For conformance class loop-blocked

the network structure has to be acyclic and for conformance class full-blocked there has to be a one-to-one correspondence between splits and joins of the same type.

**WP11 Implicit Termination** A given subprocess is terminated when there is nothing left to do, i.e., termination does not require an explicit termination activity. The goal of this pattern is to avoid having to join divergent branches into a single point of termination.

**Solution, WP11** XPDL, assuming conformance class “non-blocked”, allows for arbitrary graph-like structures. As a result it is possible to have multiple activities without input transitions (i.e., source activities) and multiple activities without output transitions (sink activities). The latter suggests direct support for WP11. Unfortunately, [28] does not clarify the semantics of XPDL in the presence of multiple source and sink activities, e.g., Do all source activities need to be executed or just one? Although XPDL does not specify the expected behavior in such cases, we give it the benefit of the doubt. Note that this illustrates that conformance is still ill-defined in [28] since it refers to syntax rather than semantics.

**WP12 MI without Synchronization** Within the context of a single case, multiple instances of an activity may be created, i.e. there is a facility for spawning off new threads of control, all of them independent of each other. The instances might be created consecutively, but they will be able to run in parallel, which distinguishes this pattern from the pattern for Arbitrary Cycles. **Example:** When booking a trip, the activity *book flight* is executed multiple times if the trip involves multiple flights.

**Solution, WP12** An activity may be refined into a subflow. The subflow may be executed synchronously or asynchronously. In case of asynchronous execution, the activity is continued after an instance of the subflow is initiated. This way it is possible to “spawn-off” subflows and thus realizing WP12.

**WP13-WP15 MI with Synchronization** A point in a workflow where a number of instances of a given activity are initiated, and these instances are later synchronized, before proceeding with the rest of the process. In WP13 the number of instances to be started/synchronized is known at design time. In WP14 the number is known at some stage during run time, but before the initiation of the instances has started. In WP15 the number of instances to be created is not known in advance: new instances

are created on demand, until no more instances are required. **Example of WP15:** When booking a trip, the activity *book flight* is executed multiple times if the trip involves multiple flights. Once all bookings are made, an invoice is sent to the client. How many bookings are made is only known at runtime through interaction with the user (or with an external process).

**Solutions, WP13-WP15** If the number of instances to be synchronized is known at design time (WP13), a simple solution is to replicate the activity as many times as it needs to be instantiated, and run the replicas in parallel. Therefore, WP13 is supported. However, it is clear that there is no direct support for WP14 and WP15 because any solution will involve explicit bookkeeping of the number of active instances. In fact in [28] is stated that “Synchronization with the initiated subflow, if required, has to be done by other means such as events, not described in this document.” when describing the functionality of asynchronous subflows. Therefore, we conclude that there is no support for WP14 and WP15. Again we refer to [6] for typical workarounds.

**WP16 Deferred Choice** A point in a process where one among several alternative branches is chosen based on information which is not necessarily available when this point is reached. This differs from the normal exclusive choice, in that the choice is not made immediately when the point is reached, but instead several alternatives are offered, and the choice between them is delayed until the occurrence of some event. **Example:** When a contract is finalized, it has to be reviewed and signed either by the director or by the operations manager, whoever is available first. Both the director and the operations manager would be notified that the contract is to be reviewed: the first one who is available will proceed with the review.

**Solution, WP16** XPDL only allows for choices resulting from conditions on transitions. Hence each choice is directly-based on workflow relevant data and it is not possible offer the choice to the environment. XPDL does not allow for the definition of states (like places in a Petri net) nor constructs like the choice construct in BPML and WSCI and the pick construct in XLANG and BPEL4WS. There is no simple work-around for this omission since it is not possible to shift the moment of decision from the end of an activity to the start of an activity. Moreover, XPDL does not allow for the specification of triggers and/or external events.

**WP17 Interleaved Parallel Routing** A set of activities is executed in an arbitrary order. Each activity in the set is executed exactly once.

The order between the activities is decided at run-time: it is not until one activity is completed that the decision on what to do next is taken. In any case, no two activities in the set can be active at the same time.

**Example:** At the end of each year, a bank executes two activities for each account: *add interest* and *charge credit card costs*. These activities can be executed in any order. However, since they both update the account, they cannot be executed at the same time.

**Solution, WP17** Since XPDL does not allow for the definition of states, it is not possible to enforce some kind of mutual exclusion. Hence there is no support for WP17. Even the work-arounds described in [6] are difficult, if not impossible, to apply.

**WP18 Milestone** A given activity can only be enabled if a certain milestone has been reached which has not yet expired. A milestone is defined as a point in the process where a given activity has finished and another activity following it has not yet started. **Example:** After having placed a purchase order, a customer can withdraw it at any time before the shipping takes place. To withdraw an order, the customer must complete a withdrawal request form, and this request must be approved by a customer service representative. The execution of the activity *approve order withdrawal* must therefore follow the activity *request withdrawal*, and can only be done if: (i) the activity *place order* is completed, and (ii) the activity *ship order* has not yet started.

**Solution, WP18** XPDL does not provide a direct support for capturing this pattern. Therefore, a work-around solution has to be used. Again it is difficult to construct solutions inspired by the ideas in [6]. Since other patterns like WP16 and WP19 are not supported, potential solutions lead to complex process definitions for simply checking the state in a parallel branch.

**WP19 Cancel Activity & WP20 Cancel Case** A cancel activity terminates a running instance of an activity, while canceling a case leads to the removal of an entire workflow instance. **Example of WP19:** A customer cancels a request for information. **Example of WP20:** A customer withdraws his/her order.

**Solutions, WP19 & WP20** XPDL does not provide explicit constructs for WP19 and WP20. The concept of exceptions seems to be related, but like many other concepts ill-defined. The only construct in XPDL

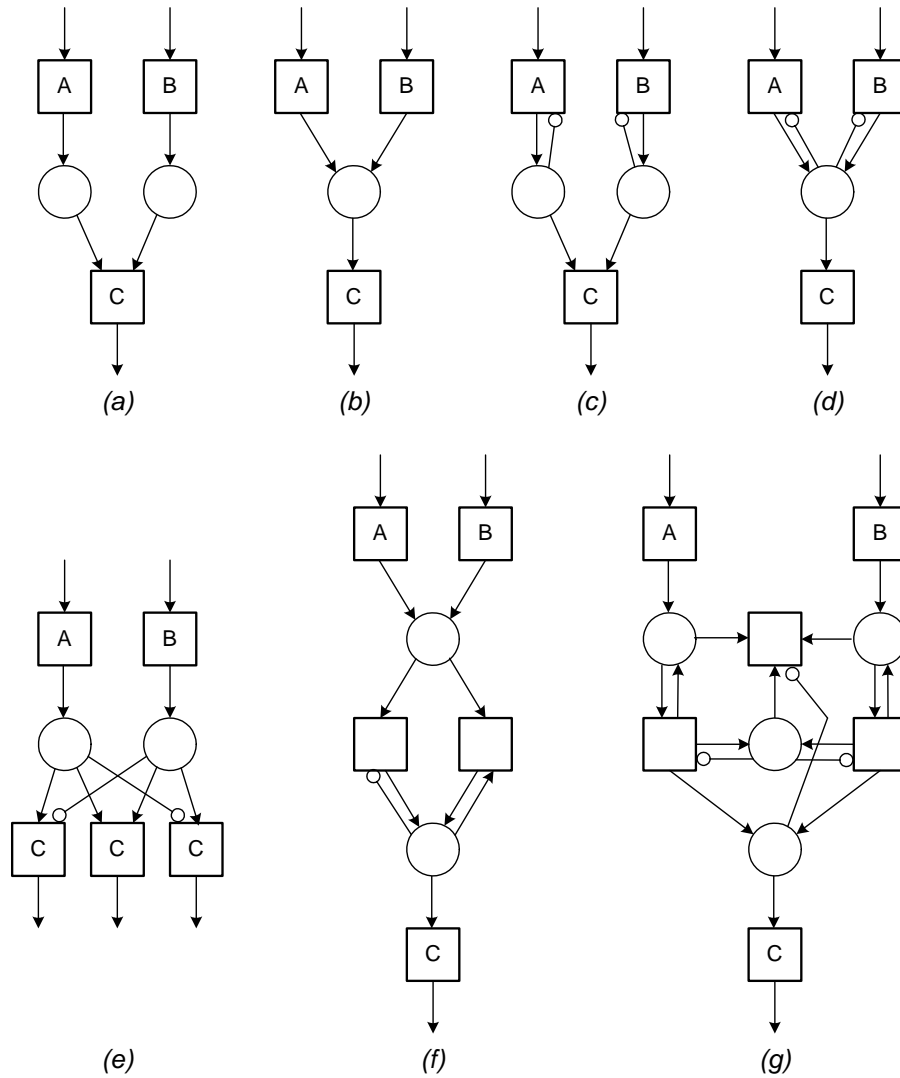
that can raise an exception is the **deadline** element. Deadlines are used to raise an exception upon the expiration of a specific period of time. A deadline can be raised synchronously or asynchronously: “If the deadline is synchronous, then the activity is terminated before flow continues on the exception path.” and “If the deadline is asynchronous, then an implicit AND-SPLIT is performed, and a new thread of processing is started on the appropriate exception transition.” [28]. An exception may trigger a transition but cannot be used to cancel activities or cases. Hence, XPDL does not support WP19 and WP20.

#### 4 Many ways to join

In the previous section, we evaluated XPDL with respect to the patterns. A more detailed analysis reveals that, not only does XPDL have problems with respect to several patterns, the semantics of many constructs is unclear. To illustrate this we focus on transition restrictions of type Join. The restriction is either set to AND or to XOR and the WfMC defines these settings as follows: “AND: Join of (all) concurrent threads within the process instance with incoming transitions to the activity: Synchronization is required. The number of threads to be synchronized might be dependent on the result of the conditions of previous AND split(s).” and “XOR: Join for alternative threads: No synchronization is required.” [28]. To demonstrate that these descriptions do not fully specify the intended behavior, Figure 1 shows seven possible interpretations each expressed in terms of a Petri net [21]. Note that Petri nets have formal semantics, and thus, Figure 1 fully specifies the behavior of each construct. Also note that we restrict ourselves to local constructs, i.e., there are no dependencies other than on the activities directly connected to the join.

The first two constructs correspond to the most straightforward interpretations of the AND-join (Figure 1(a)) and XOR-join (Figure 1(b)). In Figure 1(a), activity C always synchronizes A and B, i.e., if A is never executed, C is never executed.<sup>1</sup> In Figure 1(b), activity C is executed once for each occurrence of A and B. Although Figure 1(a) and Figure 1(b) seem to correspond to straightforward interpretations of the AND-join and XOR-join, few workflow management systems actually exhibit this behavior. The other constructs in Figure 1 show other interpretations for both the AND-join and/or XOR-join encountered in contemporary systems. Figure 1(c) shows the situation where activity A is blocked if C was not executed since the last occurrence of A. Similarly, activity B is

<sup>1</sup> Note that this is not the case in XPDL.



**Fig. 1.** Seven frequently used ways to join two flows (expressed in terms of Petri nets with inhibitor arcs [21]).

blocked if C was not executed since the last occurrence of B. Note that this construct uses two inhibitor arcs (i.e., the two connections involving a small circle). Unlike a normal directed arc in Petri net, an inhibitor arc models the requirement that a place has to be empty, i.e., A is only enabled if the input place (not shown) contains a token *and* the output place is empty. Figure 1(d) shows a similar construct but now for the XOR-join, i.e., both activity A and activity B are blocked if C was not executed since the last occurrence of A or B. The workflow management system COSA [23] uses this interpretation for the AND-join and XOR-join. Figures 1 (c) and (d) use inhibitor arcs to make sure that activity C is only enabled once. This is realized by blocking the preceding activities if needed. An alternative approach is to simply remove additional tokens. Figure 1(e) shows an approach where C synchronizes both flows if both A and B have been executed. If only one of them has been executed, there is no synchronization. Note that there are three instances of C: one for the situation where only A was executed, one for situation where both A and B have been executed, and one where only B was executed. The two inhibitor arcs make sure that the two flows are synchronized if possible. Figure 1(f) shows a similar, but slightly different, approach where simply every attempt to enable C for the second time is ignored. If C is already enabled, then the right transition will occur, otherwise the left one. Consider the scenario where A occurs twice before execution C. In Figure 1(e), C will be executed twice, while in Figure 1(f) C will be executed only once. Many systems have a behavior similar to Figure 1(e)/(f), e.g., a normal step in Staffware [24] behaves as indicated by Figure 1(f). Although widely supported, the interpretation given in Figure 1(e)/(f) is not very desirable from a modeling point of view since it introduces “race conditions”, e.g., the number of times C is executed depends on the interleaving of A, B, and C activities. Figure 1(g) gives yet another interpretation of the AND/XOR-join. C is enabled immediately after the first occurrence of A or B, but after it occurs it is blocked until the other activity has also been executed, i.e., the construct is reset once each of A, B, and C has occurred. Note that this interpretation corresponds to WP9 (Discriminator pattern).

Figure 1 shows that there are many ways to join two flows. In fact, there are many more interpretations. An example is the so-called “wait step” in Staffware [24] which only synchronizes the first time if it is put in a loop. Another example is the join in IBM’s MQSeries Workflow [13], BPEL4WS [9]), and WSFL (Web Services Flow Language, [18]) which decides whether it has to synchronize or not based on the so-called “Dead-

Path-Elimination (DPE)” [19]. Given the quote “AND: Join of (all) concurrent threads within the process instance with incoming transitions to the activity: Synchronization is required. The number of threads to be synchronized might be dependent on the result of the conditions of previous AND split(s).” in [28], the latter interpretation seems to be closest to XPDL. Unfortunately, other than IBM-influenced products and standards, no other vendors are using nor supporting this interpretation since it does not allow for Arbitrary cycles (WP10).

The dilemma of joining mixtures of alternative or parallel flows has been discussed in scientific literature. See [2] for pointers to related papers and an elaborate discussion in the context of Event-driven Process Chains (EPC’s).

The fact that there are many ways to join and that in [28] the WfMC leaves room for multiple interpretations, brings us to the issue of *conformance*. In [28] it is stated that “A product that claims conformance must generate valid, syntactically correct XPDL, and must be able to read all valid XPDL.”. Unfortunately, this quote, but also the rest of [28], does not address the issue of semantics. Note that it is rather easy to generate and read valid XPDL. The difficult part is to be able to interpret XPDL generated by another tool and execute the workflow as intended.

## 5 Conclusion

In this paper, we provided a critical evaluation of XPDL based on a set of 20 basic workflow patterns. To conclude, we compare XPDL with other standards and 15 workflow products.

Table 1 shows an evaluation of XPDL and six other standards. If a standard directly supports the pattern through one of its constructs, it is rated +. If the pattern is not *directly* supported, it is rated +/- . Any solution which results in spaghetti diagrams or coding, is considered as giving no direct support and is rated -. The rating of XPDL is as explained in this paper.

<sup>2</sup> Although the description of the AND-join suggests support for WP7, XPDL does not specify its precise behavior. In fact, for conformance class “non-blocked”, it is unclear how WP7 could be supported

<sup>3</sup> For conformance class “non-blocked”, arbitrary graph-like structures are allowed, including arbitrary cycles. For the other conformance classes this is explicitly excluded.

<sup>4</sup> For all conformance classes there may be multiple source and/or sink activities. Hence, from a syntactical point of view WP11 is supported. Unfortunately, no semantics are given for this construct.

pattern	standard						
	XPDL	UML	BPEL4WS	BPML	XLANG	WSFL	WSCI
1 (seq)	+	+	+	+	+	+	+
2 (par-spl)	+	+	+	+	+	+	+
3 (synch)	+	+	+	+	+	+	+
4 (ex-ch)	+	+	+	+	+	+	+
5 (simple-m)	+	+	+	+	+	+	+
6 (m-choice)	+	-	+	-	-	+	-
7 (sync-m)	+ <sup>2</sup>	-	+	-	-	+	-
8 (multi-m)	-	-	-	+/-	-	-	+/-
9 (disc)	-	-	-	-	-	-	-
10 (arb-c)	+ <sup>3</sup>	-	-	-	-	-	-
11 (impl-t)	+ <sup>4</sup>	-	+	+	-	+	+
12 (mi-no-s)	+	-	+	+	+	+	+
13 (mi-dt)	+	+	+	+	+	+	+
14 (mi-rt)	-	+	-	-	-	-	-
15 (mi-no)	-	-	-	-	-	-	-
16 (def-c)	-	+	+	+	+	-	+
17 (int-par)	-	-	+/-	-	-	-	-
18 (milest)	-	-	-	-	-	-	-
19 (can-a)	-	+	+	+	+	+	+
20 (can-c)	-	+	+	+	+	+	+

**Table 1.** A comparison of XPDL with other standards such as UML Activity Diagrams, BPEL4WS, BPML, XLANG, WSFL, and WSCI.

UML activity diagrams [12] are intended to model both computational and organizational processes. Increasingly, UML activity diagrams are also used for workflow modeling. Therefore, it is interesting to analyze their expressiveness using the set of basic workflow patterns as shown in the table. for more information see [10].

The recently released BPEL4WS (Business Process Execution Language for Web Services, [9]) specification builds on IBM's WSFL (Web Services Flow Language, [18]) and Microsoft's XLANG (Web Services for Business Process Design, [25]). XLANG is a block-structured language with basic control flow structures such as sequence, switch (for conditional routing), while (for looping), all (for parallel routing), and pick (for race conditions based on timing or external triggers). In contrast to XLANG, WSFL is not limited to block structures and allows for directed graphs. The graphs can be nested but need to be acyclic. Iteration is only supported through exit conditions, i.e., an activity/subprocess is iterated until its exit condition is met. The control flow part of WSFL is almost identical to the workflow language used by IBM's MQ Series Work-

flow. See [29] for more information about the evaluation of BPEL4WS, XLANG, and WSFL using the patterns.

BPML (Business Process Modeling language, [8]) is a standard developed and promoted by BPMI.org (the Business Process Management Initiative). BPMI.org is supported by several organizations, including Intalio, SAP, Sun, and Versata. The Web Service Choreography Interface (WSCI, [7]) submitted in June 2002 to the W3C by BEA Systems, BPMI.org, Commerce One, Fujitsu Limited, Intalio, IONA, Oracle Corporation, SAP AG, SeeBeyond Technology Corporation, and Sun Microsystems. There is a substantial overlap between BPML and WSCI. See [3] for more information about the evaluation of BPML and WSCI using the patterns.

In addition to comparing XPDL to other standards, it is interesting to compare XPDL with contemporary workflow management systems. Tables 2 and 3 summarize the results of the comparison of 15 workflow management systems in terms of the selected patterns. These tables are taken from [6] and have been added to compare contemporary workflow products with XPDL.

pattern	product							
	Staffware	COSA	InConcert	Eastman	FLOWer	Domino	Meteor	Mobile
1 (seq)	+	+	+	+	+	+	+	+
2 (par-spl)	+	+	+	+	+	+	+	+
3 (synch)	+	+	+	+	+	+	+	+
4 (ex-ch)	+	+	+/-	+	+	+	+	+
5 (simple-m)	+	+	+/-	+	+	+	+	+
6 (m-choice)	-	+	+/-	+/-	-	+	+	+
7 (sync-m)	-	+/-	+	+	-	+	-	-
8 (multi-m)	-	-	-	+	+/-	+/-	+	-
9 (disc)	-	-	-	+	+/-	-	+/-	+
10 (arb-c)	+	+	-	+	-	+	+	-
11 (impl-t)	+	-	+	+	-	+	-	-
12 (mi-no-s)	-	+/-	-	+	+	+/-	+	-
13 (mi-dt)	+	+	+	+	+	+	+	+
14 (mi-rt)	-	-	-	-	+	-	-	-
15 (mi-no)	-	-	-	-	+	-	-	-
16 (def-c)	-	+	-	-	+/-	-	-	-
17 (int-par)	-	+	-	-	+/-	-	-	+
18 (milest)	-	+	-	-	+/-	-	-	-
19 (can-a)	+	+	-	-	+/-	-	-	-
20 (can-c)	-	-	-	-	+/-	+	-	-

**Table 2.** The main results for Staffware, COSA, InConcert, Eastman, FLOWer, Lotus Domino Workflow, Meteor, and Mobile.

From the comparison it is clear that no tool supports all of the selected patterns. In fact, many of these tools only support a relatively small subset of the more advanced patterns (i.e., patterns 6 to 20). Specifically the limited support for the discriminator, the state-based patterns (only COSA), the synchronization of multiple instances (only FLOWer) and cancellation (esp. of activities), is worth noting.

pattern	product						
	MQSeries	Forté	Verve	Vis. WF	Changeng	I-Flow	SAP/R3
1 (seq)	+	+	+	+	+	+	+
2 (par-spl)	+	+	+	+	+	+	+
3 (synch)	+	+	+	+	+	+	+
4 (ex-ch)	+	+	+	+	+	+	+
5 (simple-m)	+	+	+	+	+	+	+
6 (m-choice)	+	+	+	+	+	+	+
7 (sync-m)	+	-	-	-	-	-	-
8 (multi-m)	-	+	+	-	-	-	-
9 (disc)	-	+	+	-	+	-	+
10 (arb-c)	-	+	+	+/-	+	+	-
11 (impl-t)	+	-	-	-	-	-	-
12 (mi-no-s)	-	+	+	+	-	+	-
13 (mi-dt)	+	+	+	+	+	+	+
14 (mi-rt)	-	-	-	-	-	-	+/-
15 (mi-no)	-	-	-	-	-	-	-
16 (def-c)	-	-	-	-	-	-	-
17 (int-par)	-	-	-	-	-	-	-
18 (milest)	-	-	-	-	-	-	-
19 (can-a)	-	-	-	-	-	-	+
20 (can-c)	-	+	+	-	+	-	+

**Table 3.** The main results for MQSeries, Forté Conductor, Verve, Visual WorkFlo, Changengine, I-Flow, and SAP/R3 Workflow.

Please apply the results summarized in tables 1, 2 and 3 with care. First of all, the organization selecting a workflow management system/-standard should focus on the patterns most relevant for the workflow processes at hand. Since support for the more advanced patterns is limited, one should focus on the patterns most needed. Second, the fact that a pattern is not directly supported by a product does not imply that it is not possible to support the construct at all. As indicated in [6], many patterns can be supported indirectly through mixtures of more basic patterns and coding. Third, the patterns reported in this paper only focus on the process perspective (i.e., control flow or routing). The other perspectives (e.g., organizational modeling) should also be taken into account.

Tables 1, 2 and 3 allow for an objective comparison of the 7 standards and 15 workflow management systems. When comparing XPDL to the 6 other standards, it is remarkable to see that XPDL seems to be less expressive than web service composition languages such as BPEL4WS and BPML. An important pattern like the Deferred choice (WP16) is supported by most standards and is vital for practical application of workflow management. Nevertheless, it is not even mentioned in [28]. Compared to the 15 workflow management systems, XPDL is not as expressive as one would expect. Many systems offer functionality (e.g., the Deferred choice and the Cancel activity patterns), not supported by XPDL. It almost seems that XPDL offers the intersection rather than the union of the functionality offered by contemporary systems. This may have been the initial goal of XPDL. However, if this is the case, two important questions need to be answered.

1. If XPDL offers the intersection rather than the union of the functionality of existing systems, then how to use XPDL in practice? Should workflow designers that want to be able to export only use a subset of the functionality offered by the system? If so, users would not be able to use powerful concepts like the Deferred choice (WP16) and the Cancel activity (WP19) patterns.
2. Why does XPDL support the Synchronizing merge (WP7) while it is only supported by a few systems. Widely-used systems like Staffware do not support this pattern, and therefore, will be unable to interpret the AND-join as indicated in [28].

Note that the issues raised cannot be solved satisfactorily. If XPDL offers the intersection of the functionality of existing systems, it is less expressive than many of the existing tools and standards. If XPDL offers the union of available functionality, it may become impossible to import a process definition into a concrete system and interpret it correctly. (Recall that no system supports all patterns.) Unfortunately, this dilemma is not really addressed by the WfMC [28]. The introduction of extended attributes (i.e., extensions of XPDL for a specific product) and conformance classes (i.e., restrictions to allow the use of specific products) are no solution and only complicate matters.

There have been several comparisons of some of the languages mentioned in this paper. These comparisons typically do not use a framework and provide an opinion rather than a structured analysis. A positive example is [22] where XPDL, BPML and BPEL4WS are compared by re-

lating the concepts used in the three languages. Unfortunately, the paper raises more questions than it answers.

Besides the dilemma that XPDL is either not expressive enough or too expressive, there is the problem of semantics. In [28] the WfMC does not give unambiguous specification of all the elements in the language. As a result, many vendors can claim to be compliant while interpreting constructs in a different way. In Section 4, we demonstrated that there are many interpretations of seemingly basic constructs like the AND-join and XOR-join. The lack of semantics restricts the application of XPDL and does not allow for a meaningful realization of the topic of conformance. As indicated before, [28] defines conformance as follows: “A product that claims conformance must generate valid, syntactically correct XPDL, and must be able to read all valid XPDL.”. Clearly, this inadequate and will not stimulate further standardization in the workflow domain. As a result, web service composition languages like BPML and BPEL4WS may take over the role of XPDL [1].

**Acknowledgments.** The author would like to thank Arthur ter Hofstede, Marlon Dumas, Petia Wohed, Bartek Kiepuszewski, and Alistair Barros, for their collaborative work on the workflow patterns.

**Disclaimer.** We, the authors and the associated institutions, assume no legal liability or responsibility for the accuracy and completeness of any information about XPDL or any of the other standards/products mentioned in this paper. However, we made all possible efforts to ensure that the results presented are, to the best of our knowledge, up-to-date and correct.

## References

1. W.M.P. van der Aalst. Don't go with the flow: Web services composition standards exposed. *IEEE Intelligent Systems*, 18(1):72–76, 2003.
2. W.M.P. van der Aalst, J. Desel, and E. Kindler. On the Semantics of EPCs: A Vicious Circle. In M. Nüttgens and F.J. Rump, editors, *Proceedings of the EPK 2002: Business Process Management using EPCs*, pages 71–80, Trier, Germany, November 2002. Gesellschaft für Informatik, Bonn.
3. W.M.P. van der Aalst, M. Dumas, A.H.M. ter Hofstede, and P. Wohed. Pattern-Based Analysis of BPML (and WSCI). QUT Technical report, FIT-TR-2002-05, Queensland University of Technology, Brisbane, 2002.
4. W.M.P. van der Aalst and K.M. van Hee. *Workflow Management: Models, Methods, and Systems*. MIT press, Cambridge, MA, 2002.
5. W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Advanced Workflow Patterns. In O. Etzion and P. Scheuermann, editors, *7th International Conference on Cooperative Information Systems (CoopIS 2000)*, volume

- 1901 of *Lecture Notes in Computer Science*, pages 18–29. Springer-Verlag, Berlin, 2000.
6. 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.
  7. A. Arkin, S. Askary, S. Fordin, and W. Jekel et al. Web Service Choreography Interface (WSCI) 1.0. Standards proposal by BEA Systems, Intalio, SAP, and Sun Microsystems, 2002.
  8. A. Arkin et al. Business Process Modeling Language (BPML), Version 1.0, 2002.
  9. F. Curbera, Y. Golland, J. Klein, F. Leymann, D. Roller, S. Thatte, and S. Weerawarana. Business Process Execution Language for Web Services, Version 1.0. Standards proposal by BEA Systems, International Business Machines Corporation, and Microsoft Corporation, 2002.
  10. M. Dumas and A.H.M. ter Hofstede. UML activity diagrams as a workflow specification language. In M. Gogolla and C. Kobryn, editors, *Proc. of the 4th Int. Conference on the Unified Modeling Language (UML01)*, volume 2185 of *LNCS*, pages 76–90, Toronto, Canada, October 2001. Springer Verlag.
  11. L. Fischer, editor. *Workflow Handbook 2001, Workflow Management Coalition*. Future Strategies, Lighthouse Point, Florida, 2001.
  12. Object Management Group. *OMG Unified Modeling Language 2.0 Proposal, Revised submission to OMG RFPs ad/00-09-01 and ad/00-09-02, Version 0.671*. OMG, <http://www.omg.com/uml/>, 2002.
  13. IBM. *IBM MQSeries Workflow - Getting Started With Buildtime*. IBM Deutschland Entwicklung GmbH, Boeblingen, Germany, 1999.
  14. S. Jablonski and C. Bussler. *Workflow Management: Modeling Concepts, Architecture, and Implementation*. International Thomson Computer Press, London, UK, 1996.
  15. B. Kiepuszewski. *Expressiveness and Suitability of Languages for Control Flow Modelling in Workflows*. PhD thesis, Queensland University of Technology, Brisbane, Australia, 2002. Available via <http://www.tm.tue.nl/it/research/patterns>.
  16. B. Kiepuszewski, A.H.M. ter Hofstede, and W.M.P. van der Aalst. Fundamentals of Control Flow in Workflows. *Acta Informatica*, 39(3):143–209, 2003.
  17. P. Lawrence, editor. *Workflow Handbook 1997, Workflow Management Coalition*. John Wiley and Sons, New York, 1997.
  18. F. Leymann. Web Services Flow Language, Version 1.0, 2001.
  19. F. Leymann and D. Roller. *Production Workflow: Concepts and Techniques*. Prentice-Hall PTR, Upper Saddle River, New Jersey, USA, 1999.
  20. D.C. Marinescu. *Internet-Based Workflow Management: Towards a Semantic Web*, volume 40 of *Wiley Series on Parallel and Distributed Computing*. Wiley-Interscience, New York, 2002.
  21. W. Reisig and G. Rozenberg, editors. *Lectures on Petri Nets I: Basic Models*, volume 1491 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1998.
  22. R. Shapiro. A Comparison of XPDL, BPML and BPEL4WS (Version 1.4). <http://xml.coverpages.org/Shapiro-XPDL.pdf>, 2002.
  23. Software-Ley. *COSA 3.0 User Manual*. Software-Ley GmbH, Pullheim, Germany, 1999.
  24. Staffware. *Staffware 2000 / GWD User Manual*. Staffware plc, Berkshire, United Kingdom, 2000.
  25. S. Thatte. XLANG Web Services for Business Process Design, 2001.
  26. WfMC. Workflow Management Coalition Terminology and Glossary (WfMC-TC-1011). Technical report, Workflow Management Coalition, Brussels, 1996.

27. WFMC. Workflow Management Coalition Workflow Standard: Interface 1 – Process Definition Interchange Process Model (WFMC-TC-1016). Technical report, Workflow Management Coalition, Lighthouse Point, Florida, USA, 1999.
28. WFMC. Workflow Management Coalition Workflow Standard: Workflow Process Definition Interface – XML Process Definition Language (XPDL) (WFMC-TC-1025). Technical report, Workflow Management Coalition, Lighthouse Point, Florida, USA, 2002.
29. P. Wohed, W.M.P. van der Aalst, M. Dumas, and A.H.M. ter Hofstede. Pattern-Based Analysis of BPEL4WS. QUT Technical report, FIT-TR-2002-04, Queensland University of Technology, Brisbane, 2002.
30. Workflow Patterns Home Page. <http://www.tm.tue.nl/it/research/patterns>.

## A XPDL Schema

The listing below shows selected parts of the XPDL Schema given in [28] relevant for this paper.

```

1 <xsd:element name="Activity">
2   <xsd:complexType>
3     <xsd:sequence>
4       <xsd:element ref="xpdl:Description" minOccurs="0"/>
5       <xsd:element ref="xpdl:Limit" minOccurs="0"/>
6       <xsd:choice>
7         <xsd:element ref="xpdl:Route"/>
8         <xsd:element ref="xpdl:Implementation"/>
9         <xsd:element ref="xpdl:BlockActivity"/>
10      </xsd:choice>
11      <xsd:element ref="xpdl:Performer" minOccurs="0"/>
12      <xsd:element ref="xpdl:StartMode" minOccurs="0"/>
13      <xsd:element ref="xpdl:FinishMode" minOccurs="0"/>
14      <xsd:element ref="xpdl:Priority" minOccurs="0"/>
15      <xsd:element ref="xpdl:Deadline" minOccurs="0"
16                maxOccurs="unbounded"/>
17      <xsd:element ref="xpdl:SimulationInformation" minOccurs="0"/>
18      <xsd:element ref="xpdl:Icon" minOccurs="0"/>
19      <xsd:element ref="xpdl:Documentation" minOccurs="0"/>
20      <xsd:element ref="xpdl:TransitionRestrictions" minOccurs="0"/>
21      <xsd:element ref="xpdl:ExtendedAttributes" minOccurs="0"/>
22    </xsd:sequence>
23    <xsd:attribute name="Id" type="xsd:NMTOKEN" use="required"/>
24    <xsd:attribute name="Name" type="xsd:string"/>
25  </xsd:complexType>

```

```
26 </xsd:element>
27 ...
28 <xsd:element name="TransitionRestriction">
29   <xsd:complexType>
30     <xsd:sequence>
31       <xsd:element ref="xpd1:Join" minOccurs="0"/>
32       <xsd:element ref="xpd1:Split" minOccurs="0"/>
33     </xsd:sequence>
34   </xsd:complexType>
35 </xsd:element> <xsd:element name="TransitionRestrictions">
36   <xsd:complexType>
37     <xsd:sequence>
38       <xsd:element ref="xpd1:TransitionRestriction" minOccurs="0"
39         maxOccurs="unbounded"/>
40     </xsd:sequence>
41   </xsd:complexType>
42 </xsd:element>
43 ...
44 <xsd:element name="Join">
45   <xsd:complexType>
46     <xsd:attribute name="Type">
47       <xsd:simpleType>
48         <xsd:restriction base="xsd:NMTOKEN">
49           <xsd:enumeration value="AND"/>
50           <xsd:enumeration value="XOR"/>
51         </xsd:restriction>
52       </xsd:simpleType>
53     </xsd:attribute>
54   </xsd:complexType>
55 </xsd:element>
56 ...
57 <xsd:element name="Split">
58   <xsd:complexType>
59     <xsd:sequence>
60       <xsd:element ref="xpd1:TransitionRefs" minOccurs="0"/>
61     </xsd:sequence>
62     <xsd:attribute name="Type">
63       <xsd:simpleType>
64         <xsd:restriction base="xsd:NMTOKEN">
65           <xsd:enumeration value="AND"/>
```

```
66         <xsd:enumeration value="XOR"/>
67     </xsd:restriction>
68 </xsd:simpleType>
69 </xsd:attribute>
70 </xsd:complexType>
71 </xsd:element>
72 ...
73 <xsd:element name="Transition">
74     <xsd:complexType>
75         <xsd:sequence>
76             <xsd:element ref="xpd1:Condition" minOccurs="0"/>
77             <xsd:element ref="xpd1:Description" minOccurs="0"/>
78             <xsd:element ref="xpd1:ExtendedAttributes" minOccurs="0"/>
79         </xsd:sequence>
80         <xsd:attribute name="Id" type="xsd:NMTOKEN" use="required"/>
81         <xsd:attribute name="From" type="xsd:NMTOKEN" use="required"/>
82         <xsd:attribute name="To" type="xsd:NMTOKEN" use="required"/>
83         <xsd:attribute name="Name" type="xsd:string"/>
84     </xsd:complexType>
85 </xsd:element>
86 ...
87 <xsd:element name="Condition">
88     <xsd:complexType mixed="true">
89         <xsd:choice minOccurs="0" maxOccurs="unbounded">
90             <xsd:element ref="xpd1:Xpression"/>
91         </xsd:choice>
92         <xsd:attribute name="Type">
93             <xsd:simpleType>
94                 <xsd:restriction base="xsd:NMTOKEN">
95                     <xsd:enumeration value="CONDITION"/>
96                     <xsd:enumeration value="OTHERWISE"/>
97                     <xsd:enumeration value="EXCEPTION"/>
98                     <xsd:enumeration value="DEFAULTEXCEPTION"/>
99                 </xsd:restriction>
100             </xsd:simpleType>
101         </xsd:attribute>
102     </xsd:complexType>
103 </xsd:element>
```