Towards Workflow Pattern Support of Event-Driven Process Chains (EPC)

Jan Mendling[1], Gustaf Neumann[1], and Markus Nüttgens[2]

[1]Vienna University of Economics and BA
Augasse 2-6, A-1090 Wien, Austria
{firstname.lastname}@wu-wien.ac.at

[2]Hamburg University of Economics and Politics
Von-Melle-Park 9, D-20146 Hamburg, Germany
nuettgens@hwp-hamburg.de

Abstract: The 20 workflow patterns proposed by van der Aalst et al. provide a comprehensive benchmark for comparing process modelling languages. In this paper, we present a workflow pattern analysis of Event-Driven Process Chains (EPCs) which is novel in its degree of detailing. Building on this analysis, we propose three extensions to EPCs in order to provide for workflow pattern support. These are the introduction of the so-called empty connector; inclusion of multiple instantiation concepts; and a cancellation construct. The latter two are inspired by YAWL. Furthermore, describe how these extensions can be represented in EPC Markup Language (EPML).

1 Introduction

The 20 workflow patterns gathered by van der Aalst, ter Hofstede, Kiepuszewski and Barros [vdAtHKB03] are well suited for analyzing different workflow languages: workflow researchers can reference to these control flow patterns in order to compare different process modelling techniques. This is of special importance considering the heterogeneity of process modelling languages (see e.g. [MNN04]). Building on the pattern analysis and on the insight that no language provides support for all patterns, van der Aalst and ter Hofstede have defined a new workflow language called YAWL [vdAtH05]. YAWL takes workflow nets [vdA97] as a starting point and adds non-petri-nets constructs in order to support each pattern (except implicit termination) in an intuitive manner.

Besides Petri nets, Event-Driven Process Chains (EPC) [KNS92] are another popular technique for business process modelling. Yet, their focus is rather related to semi-formal process documentation than formal process specification, e.g., the SAP reference model has been defined using EPC business process models [KM94]. The debate on EPC semantics (see e.g. [Ri00, NR02, vdADK02]) has recently inspired the definition of a mathematical framework for a formalization of EPCs in [Ki04]. As a consequence, we argue that workflow pattern support can also be achieved by starting with EPCs instead of Petri nets. This paper presents a detailed workflow pattern analysis of EPCs (Section 2). In particular, we highlight the non-local semantics of the XOR-join, and its implications for workflow pattern support. Furthermore, we illustrate three extensions of EPCs that are sufficient to provide for direct support of the 20 workflow patterns reported in [vdAtHKB03] (Section 3). We present our findings in an informal manner using business process models. They represent requirements for an extended version of EPCs that we refer to as yEPCs. The letter y is a tribute to YAWL and stresses workflow pattern support of yEPCs. As EPCs are frequently used for business process modelling, we expect the extension of EPCs not only to be interesting for the research community, but also useful for the modelling practice. Finally, we discuss how EPC Markup Language (EPML) can be extended in order to capture yEPCs syntactically (Section 4). After a survey on related work (Section 5), we give a conclusion and an outlook on future research (Section 6).

2 Workflow Pattern Analysis of EPCs

In this section we will consider the EPC control flow semantics of Kindler [Ki04]. They basically reflect the ideas of Nüttgens/Rump [NR02] and Keller/Nüttgens/Scheer [KNS92]. These semantics have been implemented in the simulation tool EPC Tools [CK04]. Instead of presenting them in a formal way, we discuss how EPCs can be used to model the workflow patterns (WP) presented in [vdAtHKB03]. For a formal definition refer to [Ki04].

Workflow Pattern 1: Sequence

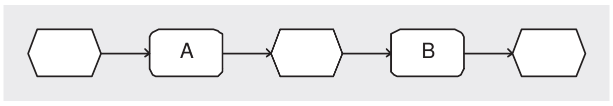

Figure 1: EPC Representation of WP1

Figure 1 shows an EPC model of workflow pattern 1 (sequence). In EPCs each activity or task is modelled as a so-called *function*. Such functions are symbolized by rounded rectangles. Functions can be connected by so-called *events* symbolized as hexagons. Events represent pre-requisites for a subsequent function, i.e., the event must have occurred before the following function may be executed. Furthermore, completed functions trigger events which may be pre-requisite for other functions. The alternation of events and functions defines a business process which also explains the name "Event-Driven" Process Chain (EPC). In Figure 1 function A triggers an event which is the pre-requisite of function B defining a sequence of activities as described by workflow pattern 1.

Workflow Pattern 2 and 3: Parallel Split and Synchronization

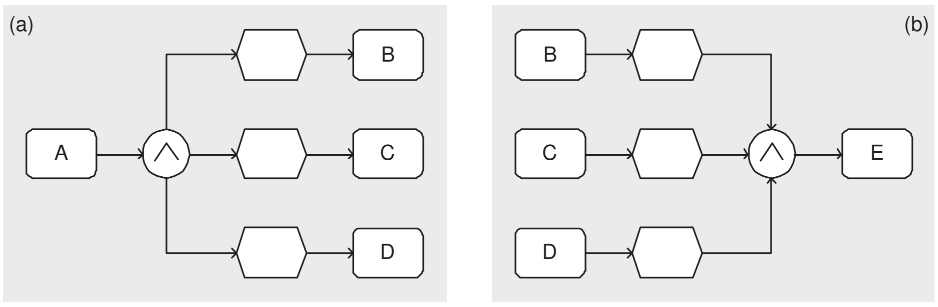

Figure 2: EPC Representation of (a) WP2 with AND split and (b) WP3 with AND join

EPCs define a restriction on the number of incoming and outgoing arcs of events and functions. Each function must have exactly one incoming and one outgoing arc, each event at most one incoming and one outgoing arc. In order to allow for complex routing of control flow so-called *connectors* are introduced. A connector may have one incoming and multiple outgoing arcs (split) or multiple incoming and one outgoing arc (join). Furthermore, each connector is mapped to one of the three connector types AND, OR, or XOR. Connectors can be used to define a partial order of functions. Figure 2 (a) illustrates the application of an AND split connector to achieve control flow behavior as defined by workflow pattern 2 (parallel split). That means after function A all the three subsequent functions B, C, and D are activated to be executed concurrently. The connector is represented by a circle. The and-symbol $\wedge$ indicates its type. Connectors have no influence on the alternation of events and functions (see e.g. [NR02, MN03a]). That means, for example, that an event is always followed by a function no matter if there are no, one, or more connectors between them. Figure 2 (b) shows the AND connector as a join. Each of the functions B, C, and D have to be completed before E can be executed. The AND join synchronizes the parallel threads of execution just as described by workflow pattern 3 (synchronization). The symbols for AND split and AND join are the same. They can only be distinguished by the cardinality of incoming and outgoing arcs.

Workflow Pattern 4 and 5: Exclusive Choice and Simple Merge

Pattern 4 (exclusive choice) describes a point in a process where a decision is made to continue with one of multiple alternative branches. This situation can be modelled with the XOR split connector of EPCs, compare Figure 3 (a). After function A has completed, a decision is taken to continue with one of functions B, C, and D. Figure 3 (b) shows the XOR join that precisely captures the semantics of pattern 5. There has been a debate on the non-local semantics of the XOR join. While Rittgen [Ri00] and Van der Aalst [vdA99] proposes a local interpretation, recent research agrees upon non-local semantics (see e.g.

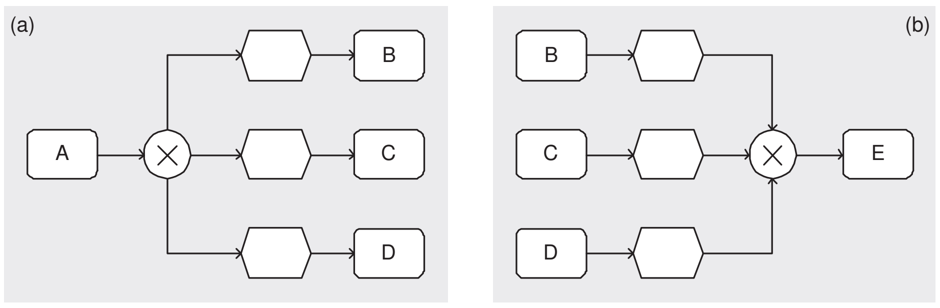

Figure 3: EPC Representation of (a) WP4 with XOR split and (b) WP5 with non-local XOR join

[NR02, Ki03, CK04]). This means that the XOR join would only allowed to continue when one of the functions B, C, and D has finished, and it is not possible that the other functions will ever be executed. Accordingly, EPC's XOR join works perfect when used in an XOR block started with an XOR split, but may block e.g. when used after an OR split depending on whether more than one branch has been activated. Regarding this non-local semantics it is similar to a synchronizing merge (see workflow pattern 7) but with the difference that it blocks when further process folders may be propagated to the XOR join.

In contrast to this, pattern 5 (simple merge) defines a merge without synchronization, but building on the assumption that the joined branches are mutually exclusive. The XOR join in YAWL [vdAtH05] can implement such such behavior with local semantics: when one of parallel activities is completed the next activity after the XOR join is started. But when the assumption does not hold, i.e., when another of the parallel activities has finished the activity after the XOR join is activated another time, and so forth. This observation allows two conclusions. First, there is a fundamental difference between the semantics of the XOR join in EPCs and YAWL: the XOR join in EPCs has non-local semantics and blocks if there are multiple paths activated; the XOR join in YAWL has local semantics and propagates each incoming process token without ever blocking. Accordingly, the YAWL XOR join can also be used to implement pattern 8 (multiple merge). Second, as the XOR join in EPCs has non-local semantics, there is no mechanism available to model workflow pattern 8 with EPCs.

Workflow Pattern 6 and 7: Multiple Choice and Synchronizing Merge

Figure 4 (a) gives an EPC model of workflow pattern 6 (multiple choice) using the OR split connector. This connector activates multiple branches based on conditions. The OR join connector depicted in Figure 4 (b) synchronizes multiple paths of execution as described in workflow pattern 7 (synchronizing merge). The OR join has both in EPCs and in YAWL non-local semantics. This means that function E can only be executed when all concurrently activated branches have completed. This is different to workflow pattern 3 (synchronization) where all branches have to complete, no matter if they have

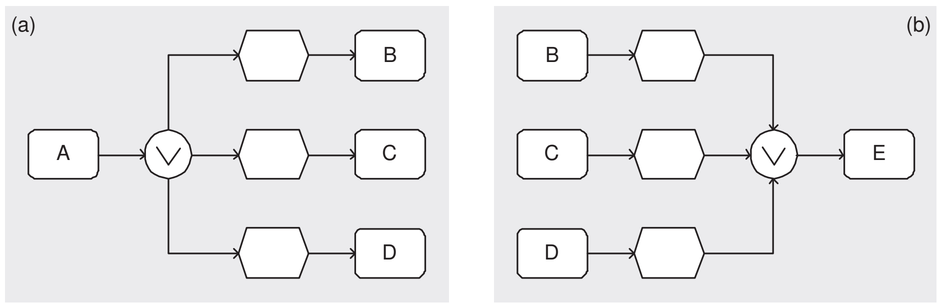

Figure 4: EPC Representation of (a) WP6 with OR split and (b) WP7 with OR join

been activated or not. Accordingly, the OR join in Figure 4 needs to consider not only if functions B, C, or D have been completed, but also if there is the chance that they can potentially be activated in the future. If this is the case, the OR join has to wait until an execution of these functions is no longer possible or until they have completed.

Workflow Pattern 10 and 11: Arbitrary Cycles and Implicit Termination

Beyond the workflow patterns 1 to 7, EPCs also provide for direct support of workflow patterns 10 and 11. Arbitrary cycles (workflow pattern 10) are explicitly allowed in EPCs. Yet, one needs to be aware that arbitrary cycles in conjunction with OR join or XOR join connectors may lead to EPC process models with so-called *unclean* semantics [Ki03]. Furthermore, it is not allowed to have cycles composed of connectors only [NR02]. Workflow pattern 11 (implicit termination) is also said to be supported by EPCs [vdAtH05]. Figure 5 gives the example of an EPC process fragment with multiple end events. This model is equivalent to a model that synchronizes these three end events with an OR join connector to only one new end event. Altogether, workflow patterns 1 to 4, 5, 6, 7, 10, and 11 are supported by EPCs. Workflow patterns 8 (multiple merge), 9 (discriminator), 12-15 (multiple instantiation), 16 (deferred choice), 17 (interleaved parallel routing), 18 (milestone), and 19-20 (cancellation) are not supported. As a consequence, business processes including control flow behavior that is related to unsupported workflow patterns cannot be represented appropriately.

3 EPC Alignment with Workflow Patterns

In order to align EPCs for direct support of workflow patterns, different modifications and extensions have to be made. In this section we propose three measures that suffice to provide for direct modelling support of all workflow patterns [vdAtHKB03] in EPCs. These measures include the introduction of the so-called empty connector; an inclusion of multiple instantiation concepts; and the introduction of a cancellation concept. This

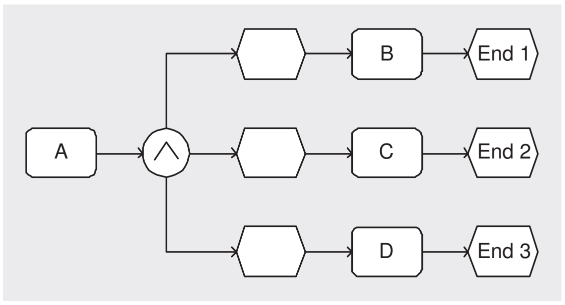

Figure 5: EPC Representation of WP11 Implicit Termination

differs from Petri net extensions that were needed to define YAWL [vdAtH05]: Petri nets also had to be extended with multiple instantiation and cancellation concepts, but they lacked advanced synchronization patterns. EPCs, in contrast, lack state-representation. Furthermore, it should be mentioned that these modifications have no impact on the validity of existing EPC models, this means that valid EPCs according to the definitions in [KNS92, NR02, Ki03] are still valid with respect to this new class of EPCs. We refer to this extended class as *yEPCs* with the letter *y* stemming from YAWL, the workflow language that inspired this research.

3.1 The Empty Connector

As mentioned above, EPCs cannot represent state-based workflow patterns. This shortcoming can be resolved by introducing a new connector type that we refer to as the empty connector. This connector is represented by a cycle, just like the other connectors, but without any symbol inside. Semantically, the empty connector represents a join or a split without imposing a rule. We will illustrate its behavior by giving EPCs that use this empty connector to model workflow patterns 16, 8, 17, and 18. In the following we interpret events similar to states. Note that the association of EPC events with states follows most research contributions on EPC formalization (see e.g. [KNS92, Ru99, Ri00, NR02]). Kindler, who uses arcs to represent states of an EPCs [Ki03], mentions that his choice was motivated rather by a straight forward presentation of his ideas than by semantical considerations. The tokens that capture the state of an EPC are called *process folders* or just *folder* [Ru99, NR02].

Workflow Pattern 16 and 8: Deferred Choice and Multiple Merge

Figure 6 (a) illustrates the application of the empty split connector to represent workflow pattern 16 (deferred choice): after function A has completed, a folder is added to the subsequent event. The empty split represents that this folder may be picked up by any of

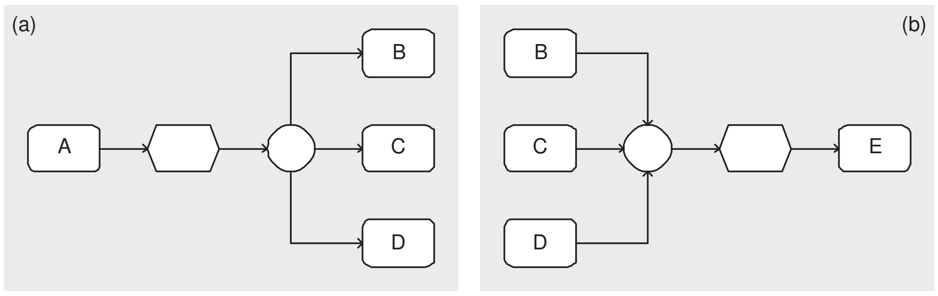

Figure 6: yEPC Representation of (a) WP16 Deferred Choice and (b) WP8 Multiple Merge

the subsequent function. Accordingly, the input pre-conditions of all three functions B, C, and D are satisfied. Yet, the first of these functions to be activated consumes the folder and by this means deactivates the other functions. Figure 6 (b) shows a process model for workflow pattern 8 (multiple merge). As we have argued in Section 2, there is no support in EPCs for the simple merge pattern due to the non-local semantics of the EPC XOR join connector. An empty join connector can be used to fix this problem. This represents that after each completion of B, C, or D a new folder is added to the pre-condition event of E. Yet, it needs to be mentioned that a design choice has to be made between a multi-set state representation as described e.g. in [NR02] and a simple set representation as specified in e.g. [Ki03]. The multi-set variant would consume further folders of C and D even if B had been executed and E not yet started. The simple set semantics would block incoming folders until the execution of E had consumed the folder on the event. The same mechanism can be used to implement workflow pattern 8 (multiple merge).

Workflow Pattern 17: Interleaved Parallel Routing

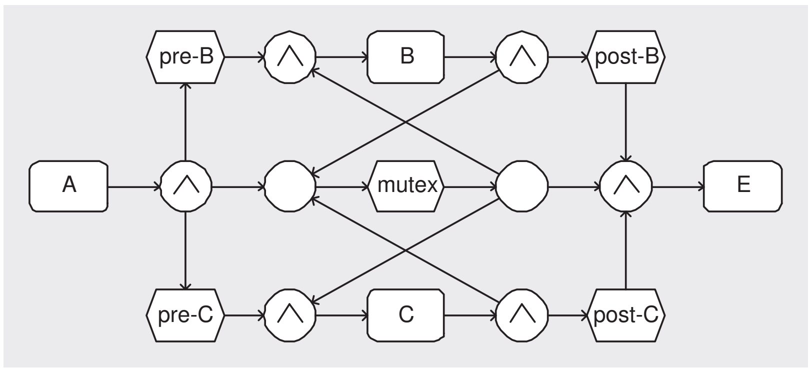

Figure 7: yEPC Representation of WP 17 Interleaved Parallel Routing

Empty connectors can also be used for other state-based workflow patterns. Figure 7 shows the process model of pattern 17 (interleaved parallel routing) following the ideas presented in [vdAtHKB03]. The event at the center of the model manages the sequential execution of functions B and C in arbitrary order. It corresponds to the "mutual exclusion place (*mutex*)" introduced in [vdAtHKB03]. The AND-split after function A adds a folder to this *mutex* event via an empty connector. The AND-joins before the functions B and C consume this folder and put it back to the mutex event afterwards. Furthermore, they consume the individual folders in *pre-B* and *pre-C*, respectively. These events control that each function of B and C is executed only once. After both have been executed, there are folders in *post-B*, *post-C*, and *mutex*. Accordingly, E can be started. In [Ro95] sequential split and join operators are proposed to describe control flow behavior of workflow pattern 17. Yet, it is no clear what the formal semantics of these operators would be when these operators are not used pairwise.

Workflow Pattern 18: Milestone

Figure 8 shows the application of empty connectors for the modelling of workflow pattern 18. The event between A and B serves as a milestone for D. This means that D can only be executed if A has completed and B has not yet started. This model exploits the newly introduced empty connector to model such behavior: if B is started before D, the milestone is expired and D can no longer be executed. If D is started before E, a folder is put to the subsequent event to D which implies that B and E can then be started. Thus, the introduction of the empty connector allows for a straight-forward modelling of workflow patterns 5 and 16 to 18.

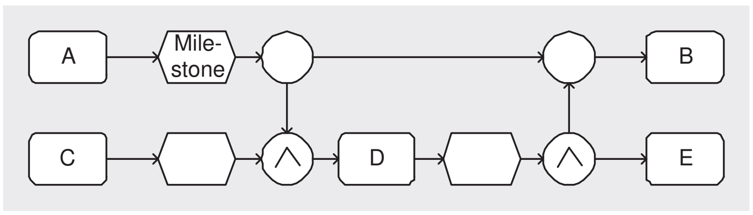

Figure 8: yEPC Representation of WP 18 Milestone

3.2 Multiple Instantiation

The lack of support for multiple instantiation has been discussed for EPCs before (see e.g. [Ro02]). In this work we stick to multiple instantiation as defined for YAWL. For related work on this topic, see e.g. [GC04] or [MSN04]. In context of multiple instantiation, it is helpful to define sub-processes in order to model complex blocks of activities that can be executed multiple times as a whole. Traditionally, there are two different kinds of sub-processes in EPCs: functions with a so-called hierarchy relation to represent the link to the sub-process [NR02, MN04] and process interfaces [KT98, MN04]. The first one, the hierarchical function, can be interpreted as a synchronous call to the sub-process. After the sub-process has completed, navigation continues with the next function subsequent to the hierarchical function. In BPML such sub-processes are modelled as a `call` activity [Ar02]. The process interface can be regarded as an asynchronous spawning off of a sub-process. There is no later synchronization when the sub-process completes. In BPML such behavior is modelled as a `spawn` activity.

Workflow Pattern 12: Multiple Instantiation without Synchronization

Figure 9 (a) shows a model fragment including a process interface. Process interfaces may be regarded as a short-hand notation for a hierarchical function that is followed by an end event. Figure 9 (b) illustrates how workflow pattern 12 (multiple instantiation without synchronization) can be modelled using a process interface. The double lines indicate that the function may be instantiated multiple times. The variables `min` and `max` define the minimum and maximum cardinality of instances that may be created. The `required` parameter specifies an integer number of instances that need to have finished in order to complete the multiple instance function. The `creation` variable may take the values `static` or `dynamic` which specify whether further instances may be created at run-time (`dynamic`) or not (`static`).

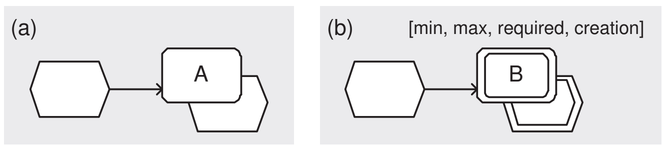

Figure 9: yEPC Representation of WP12

Workflow Pattern 13-15: Multiple Instantiation with Synchronization

Figure 10 (a) gives a model fragment of a simple function that may be instantiated multiple times (indicated by the doubled lines). Figure 10 (b) shows a hierarchical function that supports multiple instantiation. In contrast to the process interface the multiple instances are synchronized and the subsequent event is not triggered before all instances have completed.

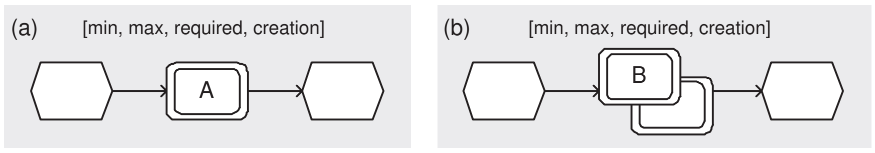
(a)
[min, max, required, creation]
A
(b)
[min, max, required, creation]
B

Figure 10: yEPC Representation of WP13-15

3.3 Cancellation

Workflow Pattern 19-20: Cancel Activity, Cancel Case

Cancellation is related to the workflow patterns 9, 19, and 20. We here adopt the concept that is used with the YAWL workflow language. Figure 11 shows the modelling notation of the cancellation concept. It specifies that when function B has completed, function A and the event is cancelled. This concept can further be used to implement workflow pattern 20, the cancellation of a whole case.

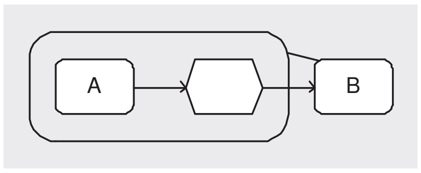

Figure 11: EPC Representation of WP19

Workflow Pattern 9: Discriminator

Beyond that, the cancellation concept can be combined with the deferred choice to model the discriminator (workflow pattern 9). Figure 12 shows a respective model fragment. The functions B, C, and D may be executed concurrently. When the first of them is completed the subsequent event is triggered. This allows function E to start. The completion of E leads to cancellation of all functions in the cancellation context that still might be active.

3.4 Requirements for yEPCs

As we have argued throughout this section, support for the 20 workflow patterns presented in [vdAtHKB03] can be achieved by extending EPCs in three different ways. First, introducing empty connectors in order to address the state-based workflow patterns. Second, multiple instantiation has to be added to EPCs. We adopted the parameters used in YAWL

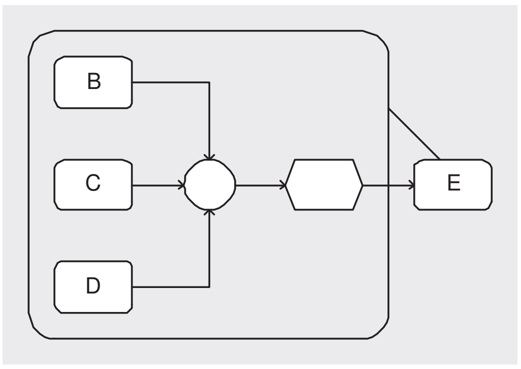

Figure 12: EPC Representation of WP9

and the doubled line notation. Multiple instances can be generated for single functions, hierarchical functions (both multiple instantiation with synchronization), and process interfaces (multiple instantiation without synchronization). Third, the cancellation concept is also adopted from YAWL. These extensions permit some conclusions on the relation of Petri nets and EPCs in general. Both had to include extensions for multiple instantiation and cancellation. In addition to this, Petri nets had to be extended with advanced synchronization concepts in order to capture the workflow patterns. On the other hand, EPCs had to be modified in order to address the state-based workflow patterns. As a consequence, yEPCs and YAWL are quite similar concerning their modelling primitives. The XOR join is the major difference between both. Yet, yEPCs still need to be formalized. The works of Kindler [Ki04] and van der Aalst and ter Hofstede [vdAtH05] are a good starting point for that.

4 EPML Alignment with yEPCs

In this section, we discuss in how far the proposed yEPC extensions may have an impact on the EPML representation. The EPC Markup Language (EPML) is an XML-based interchange format for EPC business process models proposed in [MN02, MN03b, MN04]. In this section, we particularly want to identify which syntax elements need to be added to EPML in order to represent yEPCs.

First, the introduction of the empty connector can be easily represented in the EPML schema. Figure 13 gives the example of an empty connector with an $id = 2$. The arc indicates that it follows a function with $id = 1$. Second, there are dedicated elements needed for multiple instantiation. Figure 13 gives an illustration of the required EPML elements. The `multiple` subelement indicates that the parent function or process interface can be instantiated multiple times. The four attributes capture the semantics of the parameter described in Section 3.2 and defined in [vdAtH05]. Third, the second function of Figure 13 shows how multiple `cancel` elements can be attached to a function or a pro-

```
<epml>
...
<epc epcId='1' name='example'>
     <function id='1'>
          <multiple
               minimum='3'
               maximum='6'
               required='4'
               creation='static'/>
     </function>
     <arc>
          <flow source='1' target='2'/>
     </arc>
     <empty id='2'/>
     <function id='3'>
          <cancel id='1'/>
          <cancel id='3'/>
          <cancel id='6'/>
     </function>
     ...
</epc>
</epml>
```

Figure 13: EPML Representation of multiple instantiation and cancellation

cess interface. Each cancel element carries an `id` attribute referencing the function, event, or process interface that should be cancelled. These slight extensions show that EPML can easily aligned with the syntactical requirements of yEPCs.

5 Related Work

The workflow patterns proposed by [vdAtHKB03] provide a comprehensive benchmark for comparing different process modelling languages. A short workflow pattern analysis of EPCs is also reported in [vdAtH05], yet it does not discuss the non-local semantics of EPCs XOR join. In this paper, we highlighted these semantics as a major difference between YAWL and EPCs. Accordingly, we propose the introduction of the empty connector in order to capture workflow pattern 8 (multiple merge). There is further research discussing notational extensions to EPCs. In Rittgen [Ri00] a so-called XORUND connector is proposed to partially resolve semantical problems with the XOR-join connector. Motivated by space limitations of book pages and printouts, Keller and Meinhardt introduce process interfaces to link EPC models on different pages [KM94]. We adopt process interfaces in this paper to model spawning off of sub-processes. Rosemann [Ro95] proposes the introduction of sequential split and join operators in order to capture the semantics of workflow pattern 17 (interleaved parallel routing). While the informal meaning of a pair of sequential split and join operators is clear, the formal semantics of each single operator is far from intuitive. As a consequence, we propose a state-based representation of interleaved parallel routing inspired by Petri nets. Furthermore, Rosemann introduces a connector that explicitly models a decision table and a so-called OR_1 connector to mark branches that are always executed [Ro95]. Rodenhagen presents multiple instantiation as a missing feature of EPCs [Ro02]. He proposes dedicated begin and end symbols to model that a branch of a process may be executed multiple times. Yet, this notation does not enforce that a begin symbol is followed by a matching end symbol. As a consequence, we adopt the multiple instantiation concept of YAWL that permits multiple instantiation only for single functions or sub-processes, but not for arbitrary branches of the process model.

6 Conclusion and Future Work

In this paper, we discussed current and potential future workflow pattern support of EPCs. We have presented three extensions to EPCs. These are in particular the introduction of the empty connector; the inclusion of a multiple instantiation concept for both simple functions as well as for hierarchical functions and process interfaces; and the inclusion of a cancellation concept. We refer to this extended class of EPCs as yEPCs, which is a tribute to YAWL [vdAtH05]. Furthermore, we have shown that these extensions can be easily included in EPML. In future research, we aim to define formal semantics for yEPCs and implement them in a software prototype that uses EPML as an interchange format.

References

[Ar02] Arkin, A.: Business Process Modeling Language (BPML). Specification. BPMI.org. 2002.

[CK04] Cuntz, N. und Kindler, E.: On the semantics of EPCs: Efficient calculation and simulation. In: *Proceedings of the 3rd GI Workshop on Business Process Management with Event-Driven Process Chains (EPK 2004)*. S. 7–26. 2004.

[GC04] Guabtni, A. und Charoy, F.: Multiple Instantiation in a Dynamic Workflow Environment. In: Persson, A. und Stirna, J. (Hrsg.), *Advanced Information Systems Engineering, 16th International Conference, CAiSE 2004*. volume 3084 of *Lecture Notes in Computer Science*. S. 175–188. Springer-Verlag. 2004.

[Ki03] Kindler, E.: On the semantics of EPCs: A framework for resolving the vicious circle (Extended Abstract). In: M. Nüttgens, F. J. Rump (Hrsg.), *Proc. of the 2nd GI-Workshop on Business Process Management with Event-Driven Process Chains (EPK 2003), Bamberg, Germany*. S. 7–18. 2003.

[Ki04] Kindler, E.: On the semantics of EPCs: Resolving the vicious circle. In: J. Desel and B. Pernici and M. Weske (Hrsg.), *Business Process Management, 2nd International Conference, BPM 2004*. volume 3080 of *Lecture Notes in Computer Science*. S. 82–97. Springer Verlag. 2004.

[KM94] Keller, G. und Meinhardt, S.: *SAP R/3 Analyzer. Business process reengineering based on the R/3 reference model*. SAP AG. 1994.

[KNS92] Keller, G., Nüttgens, M., und Scheer, A. W.: Semantische Prozessmodellierung auf der Grundlage "Ereignisgesteuerter Prozessketten (EPK)". Technical Report 89. Institut für Wirtschaftsinformatik Saarbrücken. Saarbrücken, Germany. 1992.

[KT98] Keller, G. und Teufel, T.: *SAP(R) R/3 Process Oriented Implementation: Iterative Process Prototyping*. Addison-Wesley. 1998.

[MN02] Mendling, J. und Nüttgens, M.: Event-Driven-Process-Chain-Markup-Language (EPML): Anforderungen zur Definition eines XML-Schemas für Ereignisgesteuerte Prozessketten (EPK). In: M. Nüttgens and F. J. Rump (Hrsg.), *Proc. of the 1st GI-Workshop on Business Process Management with Event-Driven Process Chains (EPK 2002), Trier, Germany*. S. 87–93. 2002.

[MN03a] Mendling, J. und Nüttgens, M.: EPC Modelling based on Implicit Arc Types. In: M. Godlevsky and S. W. Liddle and H. C. Mayr (Hrsg.), *Proc. of the 2nd International Conference on Information Systems Technology and its Applications (ISTA), Kharkiv, Ukraine*. volume 30 of *Lecture Notes in Informatics*. S. 131–142. 2003.

[MN03b] Mendling, J. und Nüttgens, M.: EPC Syntax Validation with XML Schema Languages. In: M. Nüttgens and F. J. Rump (Hrsg.), *Proc. of the 2nd GI-Workshop on Business Process Management with Event-Driven Process Chains (EPK 2003), Bamberg, Germany*. S. 19–30. 2003.

[MN04] Mendling, J. und Nüttgens, M.: Exchanging EPC Business Process Models with EPML. In: Nüttgens, M. und Mendling, J. (Hrsg.), *XML4BPM 2004, Proceedings of the 1st GI Workshop XML4BPM – XML Interchange Formats for Business Process Management at 7th GI Conference Modellierung 2004, Marburg Germany*. S. 61–80. http://wi.wu-wien.ac.at/˜mendling/XML4BPM/xml4bpm-2004-proceedings-epml.pdf. March 2004.

[MNN04] Mendling, J., Nüttgens, M., und Neumann, G.: A Comparison of XML Interchange Formats for Business Process Modelling. In: *Proceedings of EMISA 2004 - Information Systems in E-Business and E-Government*. LNI. 2004.

[MSN04] Mendling, J., Strembeck, M., und Neumann, G.: Extending BPEL4WS for Multiple Instantiation. In: Dadam, P. und Reichert, M. (Hrsg.), *INFORMATIK 2004 - Band 2, Proceedings of the 34th Annual Meeting of German Informatics Society (GI), Workshop Geschäftsprozessorientierte Architekturen (GPA 2004)*. volume 51 of *Lecture Notes in Informatics*. S. 524–529. Gesellschaft für Informatik. 2004.

[NR02] Nüttgens, M. und Rump, F. J.: Syntax und Semantik Ereignisgesteuerter Prozessketten (EPK). In: J. Desel and M. Weske (Hrsg.), *Promise 2002 - Proceedings of the GI-Workshop, Potsdam, Germany*. volume 21 of *Lecture Notes in Informatics*. S. 64–77. 2002.

[Ri00] Rittgen, P.: Quo vadis EPK in ARIS? Ansätze zu syntaktischen Erweiterungen und einer formalen Semantik. *WIRTSCHAFTSINFORMATIK*. 42(1):27–35. 2000.

[Ro95] Rosemann, M.: *Erstellung und Integration von Prozeßmodellen - Methodenspezifische Gestaltungsempfehlungen für die Informationsmodellierung*. PhD thesis. Westfälische Wilhelms-Universität Münster. 1995.

[Ro02] Rodenhagen, J.: Ereignisgesteuerte Prozessketten - Mulit-Instantiierungsfähigkeit und referentielle Persistenz (Event-Driven Process Chains (EPC) - Multiple Instantiation and Referential Persistence - in German). In: *Proceedings of the 1st GI Workshop on Business Process Management with Event-Driven Process Chains*. S. 95–107. 2002.

[Ru99] Rump, F. J.: *Geschäftsprozessmanagement auf der Basis ereignisgesteuerter Prozessketten - Formalisierung, Analyse und Ausführung von EPKs*. Teubner Verlag. 1999.

[vdA97] van der Aalst, W. M. P.: Verification of Workflow Nets. In: Azéma, P. und Balbo, G. (Hrsg.), *Application and Theory of Petri Nets 1997*. volume 1248 of *Lecture Notes in Computer Science*. S. 407–426. Springer Verlag. 1997.

[vdA99] van der Aalst, W. M. P.: Formalization and Verification of Event-driven Process Chains. *Information and Software Technology*. 41(10):639–650. 1999.

[vdADK02] van der Aalst, W. M. P., Desel, J., und Kindler, E.: On the semantics of EPCs: A vicious circle. In: M. Nüttgens and F. J. Rump (Hrsg.), *Proc. of the 1st GI-Workshop on Business Process Management with Event-Driven Process Chains (EPK 2002), Trier, Germany*. S. 71–79. 2002.

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

[vdAtHKB03] van der Aalst, W. M. P., ter Hofstede, A. H. M., Kiepuszewski, B., und Barros, A. P.: Workflow Patterns. *Distributed and Parallel Databases*. 14(1):5–51. July 2003.