Jan Mendling, Gustaf Neumann, Markus Nüttgens

Yet Another Event-driven Process Chain

Modelling Workflow Patterns with yEPCs

The 20 workflow patterns proposed by van der Aalst et al. provide a comprehensive benchmark for comparing process modelling languages. In this article, we discuss workflow pattern support of Event-Driven Process Chains (EPCs). 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. As both the latter are inspired by YAWL, we refer to this new class of EPCs as Yet Another Event-driven Process Chain (yEPC). Furthermore, we sketch how a transformation to YAWL can be used to specify the semantics of yEPCs.

1 Motivation

The 20 workflow patterns gathered by van der Aalst, ter Hofstede, Kiepuszewski and Barros [AHKB03] are well suited for analyzing different workflow languages: 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]). The patterns have been used to analyze several workflow and business process modelling languages in order to understand in how far they are suited to express complex behaviour in an intuitive manner. 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 [AH05]. YAWL takes workflow nets [Aa97] 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, ADK02]) 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. In this article, we define an extension to EPCs that is called Yet Another EPC (yEPC). yEPCs can be used to model all of the workflow patterns in an intuitive manner. As such they contribute to closing the gap between business process modelling with EPCs and workflow modelling with YAWL.

Before this background, the article is structured as follows. Section 2 will give a detailed workflow pattern analysis of EPCs. This shows that EPCs are able to capture several patterns, yet they fail to support state-based patterns, multiple instantiation, and cancellation patterns. Furthermore, we highlight the non-local semantics of the EPC XOR join, and its implications for workflow pattern support. In Section 3, we illustrate three extensions of EPCs that are sufficient to provide for direct support of the 20 workflow patterns. These include the empty connector, a multiple instantiation concept, and cancellation areas. Both the latter are adopted from YAWL. As yEPCs and YAWL might appear to be quite similar up to this point, we discuss sophisticated differences between the two languages in Section 4. These differences have to be reflected by a suitable transformation algorithm from yEPCs to YAWL. In Section 5, we present related research on extensions of EPCs. Section 6 closes the article and gives an outlook on future research.

2 Workflow Patterns and EPCs

EPCs are a modelling language to specify the temporal and logical relationships between activities of a business process [KNS92]. The original EPC offers the following element types: function type, event type, and connector type which can be linked via control flow arcs (see Figure 1). A function represents an activity that is executed in a process. Events represent pre- and post-conditions of functions. As a rule, functions and events have to alternate. In contrast to Petri Net-based process modelling languages, EPCs allow multiple start events and multiple end events. In EPCs there are three different kinds of connectors: AND, XOR, and OR. They may be used as either join connectors (multiple incoming, one outgoing arc) or split connectors (one incoming, multiple outgoing arcs). Even if there are connectors in between functions and events, the alternation rule must hold.

Furthermore, a distinction can be made between function-event connectors and event-function connectors. Considering this as well as the three connector types AND, XOR, and OR, and splits and joins, there are 12 possible kinds of connectors. The AND split activates all subsequent branches in concurrency while the XOR split defines a choice to activate one of multiple branches. The OR split triggers one, two or up to all of multiple branches based on conditions. In both cases of the XOR and OR split, the activation conditions are given in events subsequent to the connector. Accordingly, event-function-splits are forbidden with XOR and OR as these activation conditions do not become clear in the model. The AND join waits for all incoming branches to complete, then it propagates control to the subsequent EPC element. The semantics of the OR join have been debated as non-local – for an overview see e.g. [Ki04]. Non-locality means that the OR join synchronizes all incoming branches that are active. In order to do so, it must be aware of which branches are still active and which will never be active. In acyclic process models such synchronization can be achieved via dead-path-elimination which was also proposed for EPCs [LNS98]. Yet, cycles cannot be handled with this approach. For an approach to resolve this problem, see [Ki04]. The XOR split has also non-local semantics: if there is only one branch active (which is the expected case) it actives the subsequent EPC element. Yet, if there are multiple branches active, it synchronizes them and blocks [NR02]. EPCs offer two concepts for defining decomposition of models: hierarchical functions and process interfaces. A hierarchical function allows pointing to another EPC process that defines the behavior of the hierarchical function. The linked EPC process can be regarded as a sub-process in this context. The process interface defines a point in an EPC process where another EPC process is triggered. In contrast to a hierarchical function, this triggered process does not return control back to the process interface. In the following we illustrate how EPCs can be used to model workflow patterns [MNN05a]. For a more formal approach on EPC semantics refer to Kindler [Ki04].

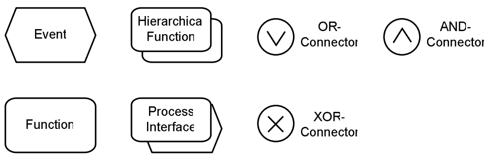

Figure 1: Symbols of the EPC notation

Workflow Pattern 1 (Sequence): Figure 2 shows an EPC model for workflow pattern 1 (sequence). In EPCs each activity or task is modelled as a so-called function symbolized by rounded rectangles. Functions can be separated via so-called events given as hexagons. As events represent pre- and post-conditions for functions the respective event must have occurred before a subsequent function can be executed. In Figure 1 (Workflow Pattern 1) function A triggers an event that is the pre-condition of function B.

Workflow Pattern 2 (Parallel Split): 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). Figure 2 (Workflow Pattern 2) illustrates how the AND split connector is applied to achieve control flow behaviour as defined by the parallel split pattern. 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. This means, for example, that an event is always followed by a function no matter if there are no, one, or more connectors between them.

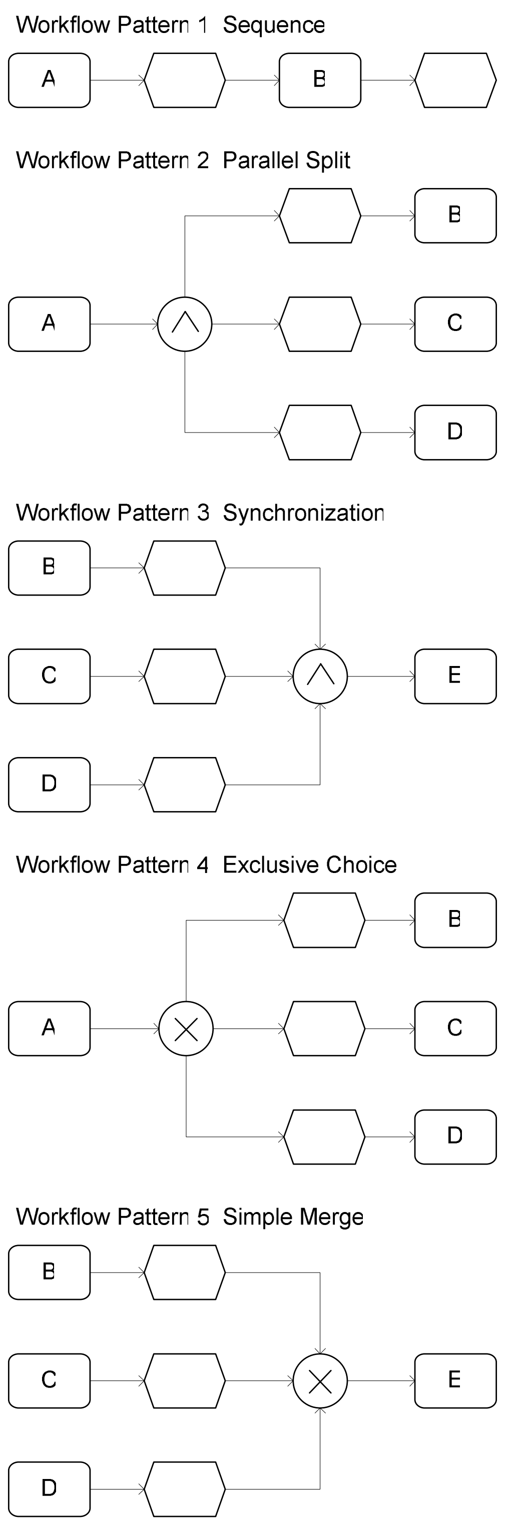
Workflow Pattern 1 Sequence
A
B
Workflow Pattern 2 Parallel Split
A
B
C
D
Workflow Pattern 3 Synchronization
B
C
D
E
Workflow Pattern 4 Exclusive Choice
A
B
C
D
Workflow Pattern 5 Simple Merge
B
C
D
E

Figure 2: Workflow Patterns 1-5 as EPC models

Workflow Pattern 3 (Synchronization): Figure 2 (Workflow Pattern 3) 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 the synchronization pattern. 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 (Exclusive Choice): 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 2 (Workflow Pattern 4). After function A has completed, a decision is taken to continue with one of functions B, C, and D.

Workflow Pattern 5 (Simple Merge): Figure 1 (Workflow Pattern 5) 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 [Aa99] proposes a local interpretation, recent research agrees upon non-local semantics (see e.g. [NR02,Ki04]). This means that the XOR join is 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 [AH05] can implement such behaviour 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.

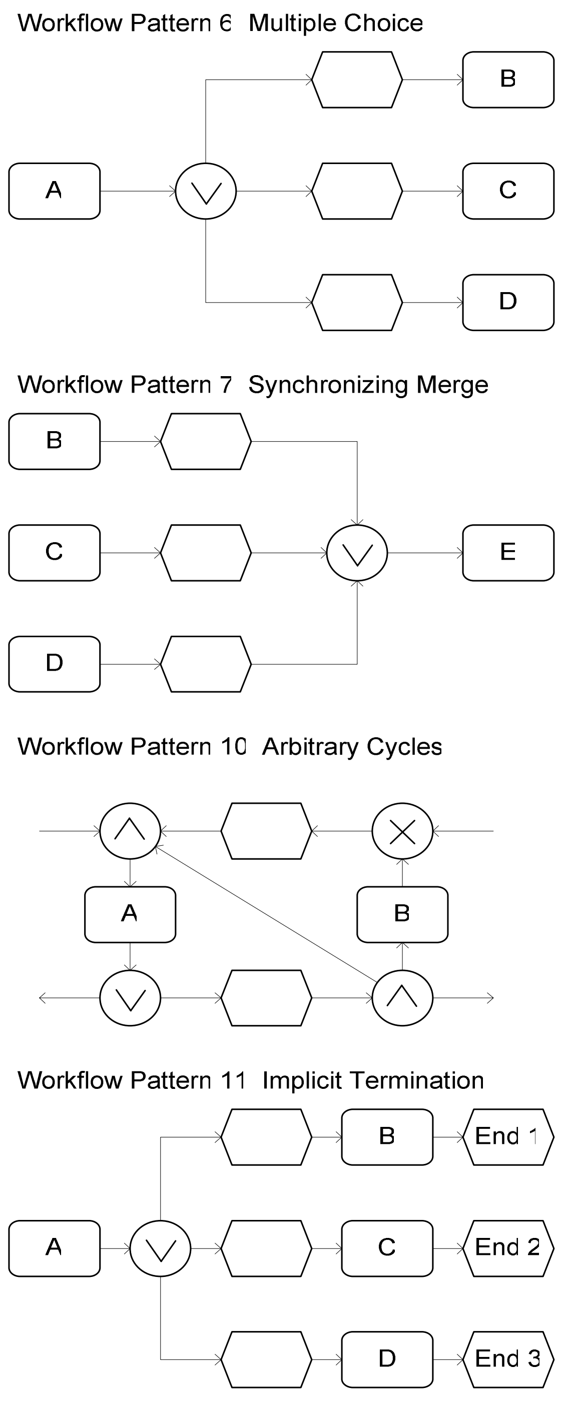
Workflow Pattern 6 Multiple Choice
A
B
C
D
Workflow Pattern 7 Synchronizing Merge
B
C
D
E
Workflow Pattern 10 Arbitrary Cycles
A
B
Workflow Pattern 11 Implicit Termination
A
B
C
D
End 1
End 2
End 3

Figure 3: Workflow Patterns 6, 7, 10, and 11 as EPC models

Workflow Pattern 6 (Multiple Choice): Figure 3 (Workflow Pattern 6) gives an EPC model for multiple choices using the OR split connector. This connector activates multiple branches based on conditions.

Workflow Pattern 7 (Synchronizing Merge): The OR join connector depicted in Figure 3 (Workflow Pattern 7) synchronizes multiple paths of execution as described in the synchronizing merge pattern. 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 been activated or not. Accordingly, the OR join in Figure 3 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 (Arbitrary Cycles): EPCs also provide for direct support of workflow pattern 10. Arbitrary cycles are explicitly allowed in EPCs. Yet, one needs to be aware that arbitrary cycles in conjunction with uncontrolled entrances via 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]. Figure 3 (Workflow Pattern 10) gives an example of a cycle with two entrance connectors at the top.

Workflow Pattern 11 (Implicit Termination): Implicit termination is also supported by EPCs [Ru99]. Figure 3 (Workflow Pattern 11) gives the example of an EPC process fragment with multiple end events. EPCs do not terminate before all activities have completed or process folders are locked in non-local XOR joins or AND joins [Ru99]. As a consequence, the model of Figure 3 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 7, 10, and 11 are supported by EPCs [MNN05a]. In the following, we introduce extensions to EPCs in order to provide for additional modelling support of workflow patterns 5 (simple merge), 8 (multiple merge), 9 (discriminator), 12-15 (multiple instantiation), 16 (deferred choice), 17 (interleaved parallel routing), 18 (milestone), and 19-20 (cancellation).

3 Workflow Patterns and yEPCs

In order to align EPCs for direct support of workflow patterns, different extensions have to be added. In this section we introduce three measures that suffice to provide for direct modelling support of all workflow patterns 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 (see Figure 4 and [MNN05b]). 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 Yet Another EPC (yEPCs) with the letter y as a reference to YAWL, the workflow language that inspired this research.

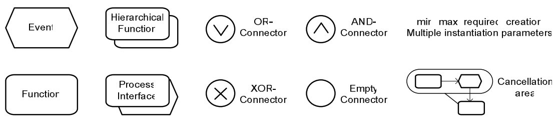

Figure 4: Symbols of the yEPC notation

3.1 The Empty Connector

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 circle, 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 behaviour by giving yEPCs 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 semantic considerations. The tokens that capture the state of an EPC are called process folders or just folder [Ru99, NR02]. In this context, empty connectors allow to put folders on an event from multiple sources (empty join) and consume folders from multiple successors of an event (empty split).

Workflow Pattern 8 (Multiple Merge): Figure 5 (Workflow Pattern 8) shows a process model for the multiple merge. As we have argued in the previous section, there is only non-local support in EPCs for the simple merge pattern due to the semantics of the EPC XOR join connector. Accordingly, the XOR join cannot be used to model the multiple merge pattern. The empty join connector can be used to fix this problem. It 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 multiset 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 5 (simple merge) with non-local semantics, yet assuming that there is only one folder that can arrive.

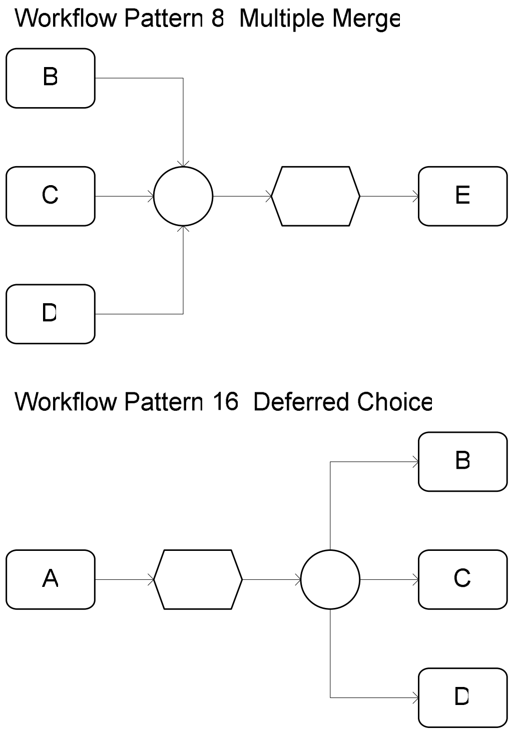
Workflow Pattern 8 Multiple Merge
B
C
D
E
Workflow Pattern 16 Deferred Choice
A
B
C
D

Figure 5: Workflow Patterns 8 and 16 as yEPC models

Workflow Pattern 16 (Deferred Choice): Figure 5 illustrates the application of the empty split connector to represent the 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 the subsequent functions.Accordingly, the input pre-conditions of all three functions B, C, and D are satisfied. Yet, the first of these functions to be

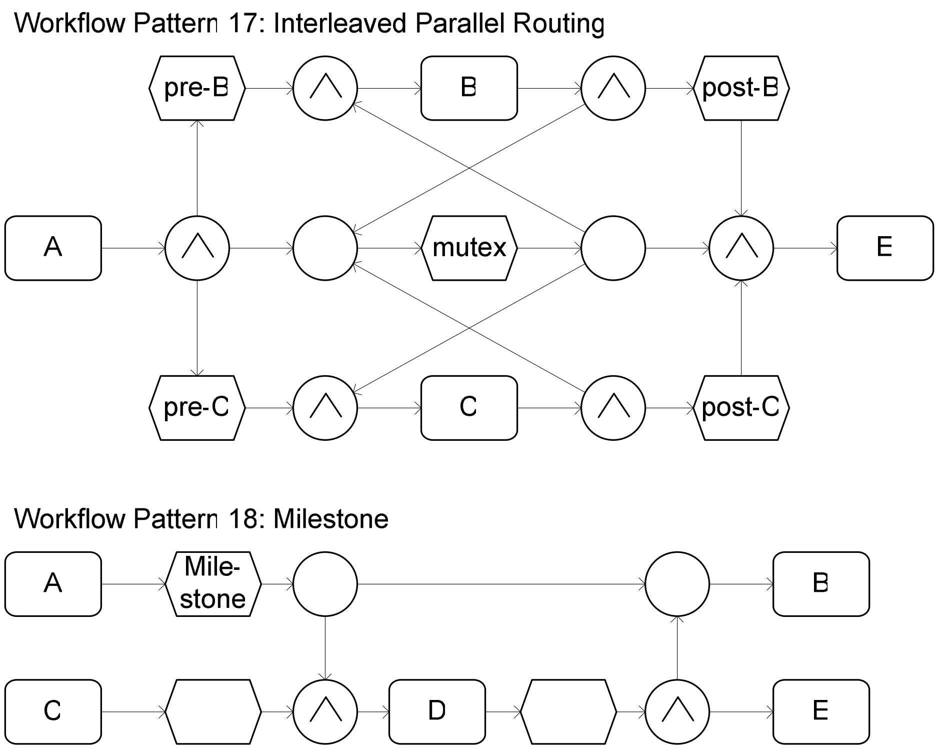
Workflow Pattern 17: Interleaved Parallel Routing
pre-B
B
post-B
A
mutex
E
pre-C
C
post-C
Workflow Pattern 18: Milestone
A
Mile-stone
B
C
D
E

Figure 6: Workflow Patterns 17 and 18 as yEPC models

activated consumes the folder and by this means deactivates the other functions.

Workflow Pattern 17 (Interleaved Parallel Routing): Empty connectors can also be used for other state-based workflow patterns. Figure 6 shows the process model of pattern 17 (interleaved parallel routing) following the ideas presented in [AHKB03]. The event at the centre 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 [AHKB03]. 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 behaviour of workflow pattern 17. Yet, it is no clear what the formal semantics of these operators would be when these operators are not used pair wise.

Workflow Pattern 18 (Milestone). Figure 6 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 behaviour: 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 8 and 16 to 18.

3.2 Multiple Instantiation

The lack of support for multiple instantiation has been discussed for EPCs before (see e.g. [Ro02]). For yEPC we adopt the respective concept from YAWL [MNN05b]. In the notation, multiple in-stantiation is represented by drawing the respective EPC symbol with double line. In this context, 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 represented by a function symbol with a second function symbol in the background [NR02, MN04] and process interfaces symbolized by a function with an event in the background [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. 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.

Workflow Pattern 12 (Multiple Instantiation without Synchronization): Figure 7 (Workflow Pattern 12) shows a model fragment including a process interface. Process interfaces can be regarded as a short-hand notation for a hierarchical function that is followed by an end event. The figure 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 have to be 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).

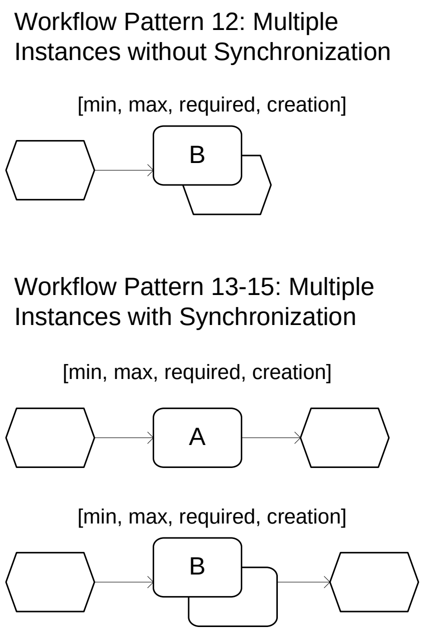
Workflow Pattern 12: Multiple Instances without Synchronization
[min, max, required, creation]
B
Workflow Pattern 13-15: Multiple Instances with Synchronization
[min, max, required, creation]
A
[min, max, required, creation]
B

Figure 7: Workflow Patterns 12-15

Workflow Pattern 13-15 (Multiple Instantiation with Synchronization): Figure 7 (Workflow Patterns 13-15) gives a model fragment of a simple function that may be instantiated multiple times (indicated by the doubled lines). Furthermore, a hierarchical function can also be specified to 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.

3.3 Cancellation

Cancellation patterns have not yet been discussed for EPCs. We adopt the concept of YAWL [MNN05b]. Cancellation areas (symbolized by a lariat) may include functions and events. The end of the lariat has to be connected with a function. When this function completes, all functions and events in the lariat are cancelled. Cancellation can be used to model workflow patterns 9, 19, and 20.

Workflow Patterns 19-20 (Cancel Activity, Cancel Case): Figure 8 (Workflow Patterns 19-20) shows the modelling notation of the cancellation concept. It specifies that when function B has completed, function A and the event are cancelled. This concept can further be used to implement workflow pattern 20, the cancellation of a whole case.

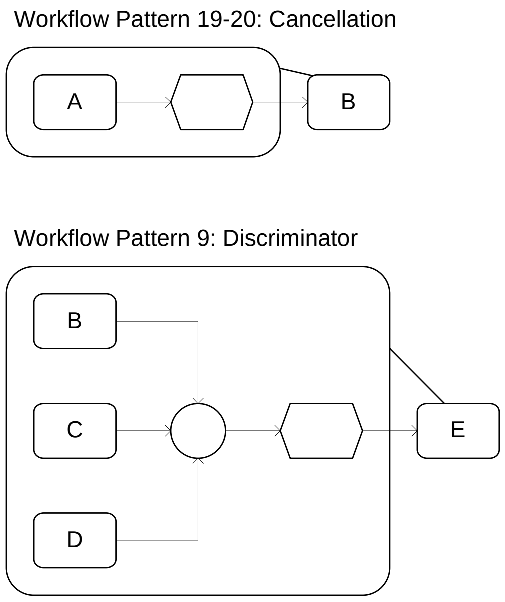
Workflow Pattern 19-20: Cancellation
A
B
Workflow Pattern 9: Discriminator
B
C
D
E

Figure 8: Workflow Patterns 9, 19-20

Workflow Pattern 9 (Discriminator): Furthermore, the cancellation concept can be combined with the deferred choice to model the discriminator. Figure 8 (Workflow Pattern 9) 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.

4 Differences between yEPC and YAWL

Both yEPC and YAWL offer quite similar primitives to model the 20 workflow patterns. Yet, there are some sophisticated differences that will be discussed in this section.

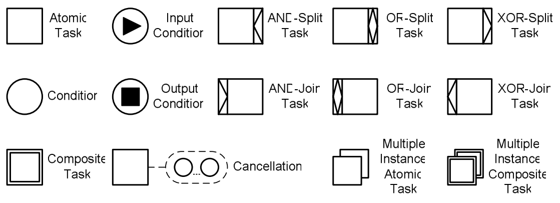

Figure 9: YAWL notation

Figure 9 gives an overview of YAWL and its notation. A YAWL process model includes exactly one input and one output condition to denote start and end of a process. Activities of a process are represented via tasks. Tasks can contain join and split rules of type AND, OR, and XOR. The XOR join has local semantics propagating all incoming tokens; the other rules have equal semantics as the respective EPC connectors. Tasks are separated by conditions which are the YAWL analogue to places in Petri nets. If two tasks are connected by an arc, the arc represents an implicit condition. Furthermore, a task can be decomposed to a sub-process. The cancellation and the multiple instantiation concept as explained before for yEPCs is adopted from YAWL.

Although yEPCs and YAWL are very similar, there are four differences which we illustrate by the help of Figure 10. The first difference is related to connectors. As connectors are independent elements in an EPC, it is allowed to build so-called connector chains, i.e. paths of two or more consecutive connectors. In Figure 9 there are three connector chains: an XOR join followed by an empty split between the start events and functions 1 and 2, and two starting with an XOR join followed by an AND split and an AND join between functions 3 to 6 and the respective end events. In YAWL splits and joins are only allowed as part of tasks. Accordingly, there is nothing like a connector chain in YAWL. The second difference stems from multiple start and end events. An EPC can include alternative start events. Multiple end events represent implicit termination: the triggering of an end event does not terminate the process as long as there is another path still active. In YAWL there is only one start condition and one end condition. The third difference is related to state representation. EPC events represent an eventuated state that can trigger a set of activities [KNS92]. Though this definition might suggest a direct mapping of events to YAWL conditions (the YAWL equivalent to places in Petri nets), there is a problem of alternative event-function and function-event connectors. In Figure 9 there is an event-function AND split after function 1 and event 1. On the other hand, the AND split after function 2 is given as a function-event split. This second alternative could be mapped element-wise to YAWL, the first one not. Accordingly, EPC events are related to states, but they do not directly match conditions in YAWL. Finally, the XOR join of EPCs has non-local semantics while the YAWL XOR join has local semantics. This means that the EPC XOR join blocks if there is more than one incoming branch active. In Figure 9 the XOR join after function 4 and 5 cannot deadlock, because both functions are exclusive due to the empty split upstream.

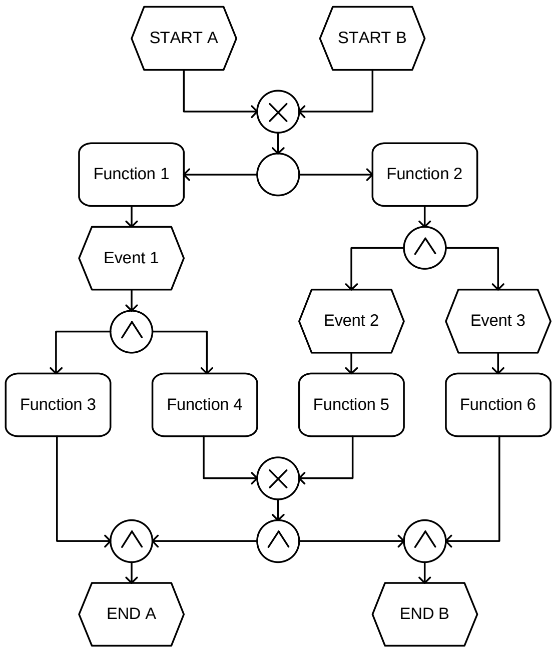

Figure 10: Example yEPC

Multiple Start and End Events: yEPC start and end events are easy to transform if there is only one start and only one end. In this case the yEPC start event maps to a YAWL input condition and the end event to a YAWL output condition. If there are multiple start events, they have to be bundled: the one YAWL input condition is followed by an empty task with an OR-split rule. Each yEPC start event is then mapped to a YAWL condition that is linked as a successor with the YAWL OR split (see Figure 11). Analogously, each of multiple yEPC end events is mapped to a YAWL condition which are all connected with an OR join of an empty task that leads to the one YAWL output condition. Note that some EPCs of the SAP Reference Model have several start events. Applying this transformation rule makes these models difficult to analyze, because $2^{|n|}$ states have to be considered with n being the amount of EPC start events. In this case, graph reduction rules could be applied in order to get compacter models. Yet, this issue is beyond the scope of this article.

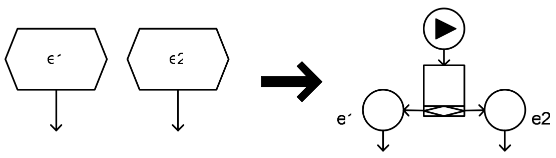

Figure 11: Mapping of Multiple Start Events

Connector Chains: Joins and splits are first class elements of yEPCs while in YAWL they are part of tasks. As a consequence, there may be the need to introduce empty tasks only to map a connector. This is in particular the case with connector chains. Figure 12 illustrates how a connector chain is transformed. If the post-event successor of a join connector is not a function, an additional empty task is required to include the join rule. If the pre-event predecessor of a split connector is not a function, an additional empty task has to include the split rule. If a join connector is followed by a split, they are combined into one empty task. Otherwise, split and joins are combined with the pre-event predecessor function or the post-event successor function, respectively.

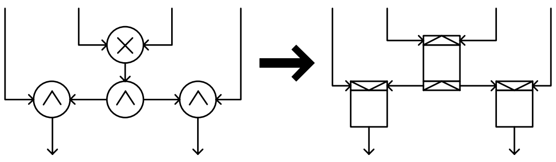

Figure 12: Mapping of Connector Chains

State Representation: As mentioned above, events cannot be identified with states directly. For the transformation the yEPC process graph can be traversed and it can be taken advantage of the fact that YAWL does not enforce an alternation of tasks and conditions. Basically, events can be ignored that are not start or end events (see Figure 13). Therefore, most states of the generated YAWL process model are associated with implicit conditions.

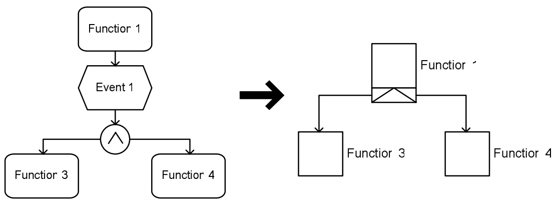

Figure 13: State Representation in yEPC and YAWL

XOR Join: Basically, in a mapping to YAWL the EPC XOR join could be mapped to an OR join with non-local semantics or an XOR join with local semantics. The latter is the better choice, because it allows a mapping back from YAWL to EPC without loss of semantics. This choice is also supported by the semantics of both XOR joins. Although the yEPC XOR join has non-local semantics leading to a deadlock if there are multiple incoming branches active and the YAWL XOR-join propagates each incoming token, the intended behaviour is the same, i.e. to continue after one of alternative branches has completed. Furthermore, in case of a deadlock in the yEPC the corresponding YAWL-net is most likely to show incorrect behaviour in terms of not being sound (for soundness of YAWL models see [AH05]).

5 Related Work

The workflow patterns proposed by [AHKB03] provide a comprehensive benchmark for comparing different process modelling languages. A short workflow pattern analysis of EPCs is also reported in [AH05], yet it does not discuss the non-local semantics of EPCs XOR join. In this article, 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 semantic problems of the XOR join connector. Motivated by space limitations of book pages and printouts, Keller and Teufel introduce process interfaces to link EPC models on different pages [KT98]. 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 Summary and Future Research

In this article, we have discussed workflow pattern support of Event-driven Process Chains (EPC). As EPCs fail to support state-based patterns as well as multiple instantiation and cancellation patterns, we have proposed yEPCs as an extension to EPCs. yEPCs introduce empty connectors, multiple instantiation parameters and cancellation areas. Therefore, yEPCs are able to support the modelling of all 20 workflow patterns in an intuitive manner. Both yEPCs and YAWL are quite similar, not only concerning the fact that both allow for comprehensive modelling of the workflow patterns[1], but also their modelling primitives are similar. Yet, there are still differences between yEPCs and YAWL: yEPCs allow multiple start and end events, yEPCs may include connector chains, state representation of yEPCs needs further investigation, and the XOR joins of both languages have different semantics. In future research, we aim to define a formal mapping from yEPCs to YAWL. This will be implemented as a transformation program from EPC Markup Language (EPML) [MN05] to the XML format of YAWL. With this transformation program, YAWL analysis tools will be accessible for EPC models.

References

[Aa97] van der Aalst,W. M. P.: Verification ofWorkflow Nets. In: Azéma, P.; Balbo, G., eds.: Application and Theory of Petri Nets 1997. volume 1248 of Lecture Notes in Computer Science. pp. 407–426. 1997.

[Aa99] van der Aalst, W.M.P.: Formalization and Verification of Event-driven Process Chains. Information and Software Technology 41 (1999) 639-650.

[ADK02] van der Aalst, W. M. P., Desel, J., und Kindler, E.: On the semantics of EPCs: A vicious circle. In: M. Nüttgens; F. J. Rump, eds.: Proc. of the 1st GI-Workshop on Business Process Management with Event-Driven Process Chains (EPK 2002), Trier, Germany. pp. 71–79. 2002.

[AH05] van der Aalst, W. M. P.; ter Hofstede, A. H. M.: YAWL: Yet Another Workflow Language. Information Systems. 30(4):245–275. 2005.

[AHKB03] van der Aalst,W. M. P.; ter Hofstede, A. H. M.; Kiepuszewski, B.; Barros, A. P.: Workflow Patterns. Distributed and Parallel Databases. 14(1):5–51. July 2003.

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

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

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

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

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

[LNS98] P. Langner, C. Schneider, and J. Wehler. Petri Net Based Certification of Event driven Process Chains. In J. Desel; M. Silva, eds.: Application and Theory of Petri Nets, volume 1420 of Lecture Notes in Computer Science, pp. 286-305, 1998.

[MN04] Mendling, J.; Nüttgens, M.: Exchanging EPC Business Process Models with EPML. In: Nüttgens, M.; Mendling, J., eds.: Proceedings of the 1st GI Workshop XML4BPM – XML Interchange Formats for Business Process Management at 7th GI Conference Modellierung 2004, Marburg Germany. pp. 61–80. March 2004.

[MN05] J. Mendling; M. Nüttgens. EPC Markup Language (EPML) - An XML-Based Interchange Format for Event-Driven Process Chains (EPC). Technical Report JM-2005-03-10, Vienna University of Economics and Business Administration, Austria, 2005.

[MNN04] Mendling, J.; Neumann, G.; Nüttgens, M.: 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.

[1] Note that YAWL does not support the implicit termination pattern.

[MNN05a] Mendling, J.; Neumann, G.; Nüttgens, M.: Towards Workflow Pattern Support of Event-Driven Process Chains (EPC). In: Nüttgens, M.; Mendling, J., eds.: Proc. of the 2nd GI Workshop XML4BPM - XML for Business Process Management at BTW 2005, Karlsruhe, Germany, pp. 23-38, March 2005.

[MNN05b] Mendling, J.; Neumann, G.; Nüttgens, M.: Yet Another Event-Driven Process Chain. In: W.M.P. van der Aalst et al.: Proceedings of the 3rd International Conference on Business Process Management (BPM 2005), volume 3649 of Lecture Notes in Computer Science, Nancy, France, September 2005, pp. 428-433.

[NR02] Nüttgens, M.; Rump, F. J.: Syntax und Semantik Ereignisgesteuerter Prozessketten (EPK). In: J. Desel; M.Weske, eds.: Promise 2002 - Proceedings of the GI-Workshop, Potsdam, Germany. volume 21 of Lecture Notes in Informatics. pp. 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.

[Ro02] Rodenhagen, J.: Ereignisgesteuerte Prozessketten - Mulit-Instantiierungsfähigkeit und referentielle Persistenz. In: M. Nüttgens, F. J. Rump, eds.: Proc. of the 1st GI Workshop on Business Process Management with Event-Driven Process Chains (EPK 2002). Trier, Germany, pp. 95–107. 2002.

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

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

Jan Mendling

Information Systems and New Media
Vienna University of Economics and Business Administration
Augasse 2-6
A-1090 Vienna
Austria
jan.mendling@wu-wien.ac.at

Prof. Dr. Gustaf Neumann

Information Systems and New Media
Vienna University of Economics and Business Administration
Augasse 2-6
A-1090 Vienna
Austria
neumann@wu-wien.ac.at

Prof. Dr. Markus Nüttgens

Business Information Systems
University of Hamburg
Von-Melle-Park 9
D-20146 Hamburg
Germany
markus.nuettgens@wiso.uni-hamburg.de