Yet Another Event-Driven Process Chain

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

[1] Vienna University of Economics and Business Administration, Austria
{firstname.lastname}@wu-wien.ac.at
[2] University of Hamburg, Germany
nuettgens@hwp-hamburg.de

Abstract. The 20 workflow patterns proposed by Van der Aalst et al. provide a comprehensive benchmark for comparing control flow aspects of process modelling languages. In this paper, we present a novel class of Event-Driven Process Chains (EPCs) that is able to capture all of these patterns. This class is called "yet another" EPC as a tribute to YAWL that inspired this research. yEPCs extend EPCs by the introduction of the so-called empty connector; inclusion of multiple instantiation concepts; and a cancellation construct. Furthermore, we illustrate how yEPCs can be used to model some of the workflow patterns.

1 Introduction

The 20 workflow patterns gathered by Van der Aalst, ter Hofstede, Kiepuszewski and Barros [1] are well suited for analyzing different workflow languages: workflow researchers can refer to these patterns in order to compare different process modelling techniques. This is of special importance considering the heterogeneity of process modelling languages (see e.g. [2]). Building 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 [3]. YAWL takes workflow nets as a starting point and adds non-petri-nets constructs in order to support each pattern in an intuitive manner (except implicit termination).

Besides Petri nets, Event-Driven Process Chains (EPC) [4] are another popular technique for business process modelling. Yet, their focus is rather related to semi-formal process documentation than formal process specification. The debate on EPC semantics has recently inspired the definition of a mathematical framework for a formalization of EPCs in [5]. As a consequence, we argue that workflow pattern support can also be achieved by starting with EPCs instead of Petri nets. This paper presents an extension to EPCs that is called yEPCs. In Section 2 we introduce EPCs and yEPCs. yEPCs include three extensions to EPCs that are sufficient to provide for direct support of the 20 workflow patterns reported in [1]. In Section 3 we discuss in detail how workflow patterns can be expressed with yEPCs. In particular, we highlight the non-local semantics of the XOR join, and its implications for workflow pattern support. After a survey on related work (Section 4), we give a conclusion and an outlook on future research (Section 5). An extended version of this paper is available as [6].

W.M.P. van der Aalst et al. (Eds.): BPM 2005, LNCS 3649, pp. 428–433, 2005.

2 Yet Another Event-Driven Process Chain (yEPC)

EPCs are introduced as a modelling concept to represent temporal and logical dependencies in business processes [4]. Elements of EPCs may be of *function type* (active elements), *event type* (passive elements), or of one of the three *connector types* AND, OR, or XOR. These objects are linked via control flow arcs. Connectors may be split or join operators, starting either with function(s) or event(s). In EPCs both OR join and XOR join have non-local semantics (cf. [5,7]). Concerning the XOR join, this implies that it blocks when there is one incoming branch finished and another still active. For a formal discussion of these semantics refer to Kindler [5]. Furthermore, *process interfaces* and *hierarchical functions* (see e.g. [7,8]) can be used to link different EPC models. A hierarchical function can be regarded as a synchronous call to a 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 a sub-process completes. For more on EPC sub-processes refer to [7].

Figure 1 illustrates the syntax elements of Yet Another Event-Driven Process Chain (yEPC). This extension of EPCs is motivated by incomplete workflow pattern support of EPCs. yEPCs reflect three measures that suffice to provide for direct support of all workflow patterns. These measures include the introduction of the so-called empty connector; an inclusion of a general multiple instantiation concept; and the introduction of a cancellation concept. The EPC extensions differ from Petri net extensions that were needed to define YAWL: Petri nets also had to be extended with multiple instantiation and cancellation concepts, but they lacked advanced synchronization patterns. EPCs, in contrast, miss support for state-based patterns. It should be mentioned that yEPC extensions have no impact on the validity of existing EPC models: this means valid EPCs according to the definitions in [7] are still valid with respect to this new class of EPCs.

As mentioned above, EPCs cannot explicitly 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. Also the same syntax rules as for other connectors hold. We follow control flow semantics as defined by Kindler [5], this means process folders (the EPC analogue to tokens

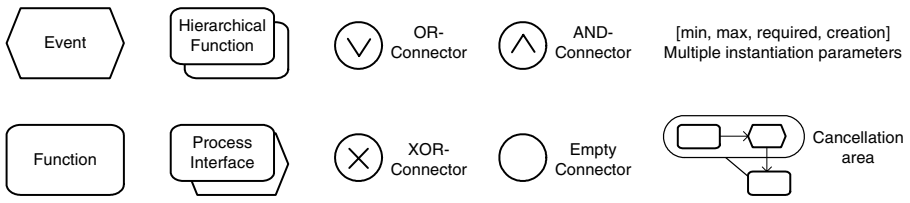

Fig. 1. yEPC Symbols

of Petri nets) are placed on arcs. The empty split then has to be interpreted as a hyperarc e.g. from the event before the empty split to the functions subsequent to it; the empty join analogously as a hyperarc from e.g. multiple functions before it to its subsequent event. Consider an event that is followed by an empty split linking to multiple functions. The empty split allows all subsequent functions to pick up the event. As a consequence, there is a run between the functions: the first function to consume the event causes the other functions to be no more active. This split semantics match the deferred choice pattern. Consider the other case of an empty join with multiple input events. The subsequent function is activated when one of these events has been reached. This behavior matched the multiple merge pattern. We will explain in Section 3 why such semantics are needed as an EPC extension.

The lack of EPC support for *multiple instantiation* has been discussed before (see e.g. [9]). In yEPCs we stick to multiple instantiation as defined for YAWL. YAWL defines a quadruple of parameters that control multiple instantiation. The parameters `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 multiple instantiation. The `creation` parameter may take the values `static` or `dynamic` which specify whether further instances may be created at run-time (`dynamic`) or not (`static`). In the 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. Accordingly, multiple instantiation parameters can be specified for functions as well as for hierarchical functions and process interfaces.

Cancellation patterns have not yet been discussed for EPCs. We adopt the concept of YAWL. Cancellation areas (symbolized by a lariat) may include functions and events. The end of the lariat has to be connected to a function. When this function completes, all functions and events in the lariat are cancelled.

3 Workflow Pattern Analysis of EPCs

In this section we will consider the EPC control flow semantics of Kindler [5] which reflect the ideas of [4,7]. For multiple instantiation and cancellation the concepts from YAWL are adopted. In the following we illustrate workflow patterns (WP) 4,5, and 17 and their yEPC representation. A full workflow pattern analysis can be found in [10]. We will speak of EPCs each time we make a statement that holds for both yEPCs and EPCs. Otherwise, we will explicitly refer to yEPCs when we present concepts that are not included in EPCs.

WP 4 (Exclusive Choice) and 5 (Simple Merge): WP 4 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. There has been a debate on the non-local semantics of the XOR join. While Rittgen [11] and Van der Aalst [12] proposes a local interpretation, recent research agrees upon non-local semantics (see e.g. [5,7]). This means that the XOR join is only allowed to continue if exactly one of the preceding functions

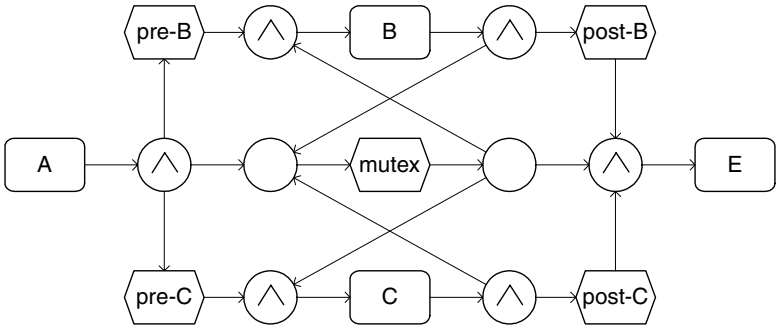

Fig. 2. yEPC Model for WP 17 Interleaved Parallel Routing

have 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. Due to these non-local semantics it is similar to a synchronizing merge but with the difference that it blocks when further process folders may be propagated to the XOR join.

In contrast to this, WP 5 defines a simple merge without synchronization, but building on the assumption that the joined branches are mutually exclusive. The XOR join in YAWL [3] can implement 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 WP 8 (multiple merge). Second, as the XOR join in EPCs has non-local semantics, it cannot be used to model WP 8. Hence, yEPCs use the empty connector for WP 8.

WP 17 (Interleaved Parallel Routing): Empty connectors can be used for state-based patterns. Figure 2 shows the process model of WP 17 following the ideas presented in [1]. 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 [1]. 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 [13] sequential split and

join operators are proposed to describe control flow behavior of WP 17. Yet, it is no clear what the semantics of these operators are when not used pairwise.

Altogether, WP 1 to 7, 10, and 11 are supported by EPCs. In contrast, yEPCs provides additional modelling support of WP 8 (multiple merge), 9 (discriminator), 12-15 (multiple instantiation), 16 (deferred choice), 17 (interleaved parallel routing), 18 (milestone), and 19-20 (cancellation). As a consequence, business processes including control flow behavior that is related to previously unsupported workflow patterns can now be represented appropriately using yEPCs.

4 Related Work

The workflow patterns proposed by [1] provide a comprehensive benchmark for comparing different process modelling languages. A short workflow pattern analysis of EPCs is also reported in [3], 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 [11] a so-called XORUND connector is proposed to partially resolve semantical 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 [8]. We adopt process interfaces in this paper to model spawning off of sub-processes. Rosemann [13] 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 [13]. Rodenhagen presents multiple instantiation as a missing feature of EPCs [9]. 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 concept of YAWL that permits multiple instantiation only for single functions or sub-processes, but not for arbitrary branches of the process model.

5 Conclusion and Future Work

In this paper, we presented a novel class of EPCs called yEPCs that is able to capture all 20 workflow patterns. Basically, yEPCs introduce three extensions to EPCs: the introduction of the empty connector; the inclusion of a multiple instantiation concept; and the inclusion of a cancellation concept. These extensions permit some conclusions on the relation of Petri nets and EPCs in general.

Towards workflow pattern support, both include extensions for multiple instantiation and cancellation. In addition, Petri nets had to be extended with advanced synchronization concepts. On the other hand, EPCs had to be modified to address the state-based patterns. As a consequence, yEPCs and YAWL are quite similar concerning their modelling primitives. The XOR join is the major difference between both. In future research, we aim to implement a transformation between yEPCs available in EPML format and the interchange format of YAWL.

References

1. van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P.: Workflow Patterns. Distributed and Parallel Databases **14** (2003) 5–51
2. Mendling, J., Neumann, G., Nüttgens, M.: A Comparison of XML Interchange Formats for Business Process Modelling. In Feltz, F., Oberweis, A., Otjacques, B., eds.: Proceedings of EMISA 2004 - Information Systems in E-Business and E-Government. Volume 56 of Lecture Notes in Informatics. (2004)
3. van der Aalst, W.M.P., ter Hofstede, A.H.M.: YAWL: Yet Another Workflow Language. Information Systems **30** (2005) 245–275
4. Keller, G., Nüttgens, M., Scheer, A.W.: Semantische Prozessmodellierung auf der Grundlage "Ereignisgesteuerter Prozessketten (EPK)". Heft 89, Institut für Wirtschaftsinformatik, Saarbrücken, Germany (1992)
5. Kindler, E.: On the semantics of EPCs: Resolving the vicious circle. In J. Desel and B. Pernici and M. Weske, ed.: Business Process Management, 2nd International Conference, BPM 2004. Volume 3080 of Lecture Notes in Computer Science., Springer Verlag (2004) 82–97
6. Mendling, J., Neumann, G., Nüttgens, M.: Yet Another Event-Driven Process Chain (Extended Version). Technical Report JM-2005-05-27, Vienna University of Economics and Business Administration, Austria (2005)
7. Nüttgens, M., Rump, F.J.: Syntax und Semantik Ereignisgesteuerter Prozessketten (EPK). In J. Desel and M. Weske, ed.: Proceedings of Promise 2002, Potsdam, Germany. Volume 21 of Lecture Notes in Informatics. (2002) 64–77
8. Keller, G., Teufel, T.: SAP(R) R/3 Process Oriented Implementation: Iterative Process Prototyping. Addison-Wesley (1998)
9. Rodenhagen, J.: Ereignisgesteuerte Prozessketten - Multi-Instantiierungsfähigkeit und referentielle Persistenz. In: Proceedings of the 1st GI Workshop on Business Process Management with Event-Driven Process Chains. (2002) 95–107
10. Mendling, J., Neumann, G., Nüttgens, M.: Towards Workflow Pattern Support of Event-Driven Process Chains (EPC). In M. Nüttgens and J. Mendling, ed.: Proc. of the 2nd Workshop XML4BPM 2005, Karlsruhe, Germany. (2005) 23–38
11. Rittgen, P.: Quo vadis EPK in ARIS? Ansätze zu syntaktischen Erweiterungen und einer formalen Semantik. WIRTSCHAFTSINFORMATIK **42** (2000) 27–35
12. van der Aalst, W.M.P.: Formalization and Verification of Event-driven Process Chains. Information and Software Technology **41** (1999) 639–650
13. Rosemann, M.: Erstellung und Integration von Prozeßmodellen - Methodenspezifische Gestaltungsempfehlungen für die Informationsmodellierung. PhD thesis, Westfälische Wilhelms-Universität Münster (1995)