On the semantics of EPCs: A vicious circle

Wil van der Aalst
Eindhoven University of Technology
w.m.p.v.d.aalst@tm.tue.nl

Jörg Desel
Katholische Universität Eichstätt-Ingolstadt
joerg.desel@ku-eichstaett.de

Ekkart Kindler
Universität Paderborn
kindler@upb.de

Abstract: Recently, Nüttgens and Rump proposed a formal semantics for Event driven Process Chains (EPCs), which should be fully compliant with the informal semantics of EPCs. But, their semantics has a severe flaw. This flaw reveals that there is a fundamental problem with the informal semantics of EPCs. Here, we pin-point the cause of this problem, we show that there is no sound formal semantics for EPCs that is fully compliant with the informal semantics, and we discuss some consequences.

1 Introduction

About ten years ago, *Event driven Process Chains* (*EPCs*) have been introduced for modelling business processes [KNS92]. Ever since, there have been different proposals of formal semantics for EPCs. Most of these semantics, however, consider only a fragment of EPCs or do not fully comply with the informal semantics of EPCs. Recently, Nüttgens and Rump proposed a formal semantics for EPCs [Rum99, NR02], and they claimed that the deficiencies of earlier formalizations have now been overcome.

The semantics of Nüttgens and Rump, however, has several insufficiencies, from a technical point of view, from a conceptual point of view, as well as from a practical point of view. These insufficiencies will be discussed in this paper. Moreover, we will show that these insufficiencies are inherent to the informal semantics of EPCs, to the effect that there cannot be a formal semantics of EPCs that is fully compliant with the informal semantics. Therefore, any formal semantics for EPCs will impose some restrictions on EPCs or will deviate form the informal semantics to some extend. Which restrictions or deviations are adequate, is a matter to be discussed. In this paper, we provide some input to such a discussion.

2 The informal semantics of EPCs

We start with a brief discussion of the informal semantics of EPCs, where we focus on one speciality of the semantics of EPCs, which we call *non-locality*. Figure 1 shows a simple example of an EPC. The dynamic behaviour of the EPC is best illustrated by *process folders*, which are propagated between the different nodes of the EPC along its control flow arcs. The *connectors*, which are represented as circles, may join and split process folders. This way, the connectors define the routing and the synchronization of process folders. For our example, we assume that, initially, there is one process folder on each of the two events Start1 and Start2.

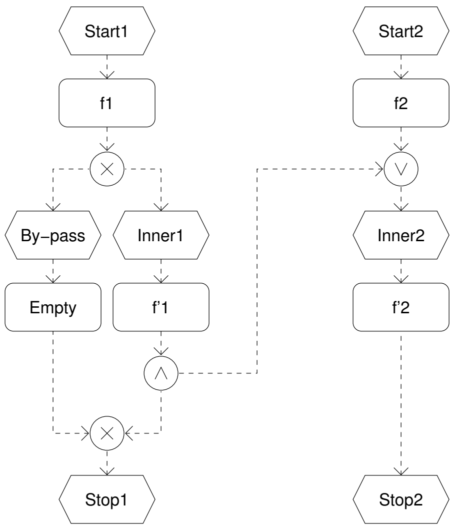

Figure 1: A simple EPC

First, we discuss what happens to the process folder on Start1: This process folder is passed to function f1. At the XOR-split connector below f1, the process folder is either propagated to the By-pass event or to the Inner1 event. If the process folder is propagated to the By-pass event, it is further propagated to the Empty function, and then passed on to the Stop1 event via the XOR-join connector. If the folder is passed to the Inner1 event, it is further propagated to the function f'1 and then reaches the AND-split connector. At the AND-split the process folder is duplicated. The two copies are propagated via the two outgoing arcs. One process folder is propagated to the XOR-join, the other is propagated to the OR-join on the right-hand side.

Second, we discuss what happens to the process folder on Start2: This process folder is propagated to function f2. What happens at the OR-join below function f2 depends on the behaviour of the left-hand part of the EPC. If there is the possibility that a process folder will arrive from the left-hand part, the OR-join delays the propagation of the process folder. In our example, this is the case as long as there is a process folder on Start1, f1, Inner1, or f'1. If no process folder can arrive from the left-hand part anymore, the OR-join propagates the folder from f2 to Inner2. In our example, this is the case, once the process folder has by-passed the AND-split. If eventually a process folder from the left-hand part arrives at the OR-join, the process folders from both incoming arcs are merged and a single process folder is propagated to the Inner2 event. From the Inner2 event, the process folder is propagated down to f'2 and Stop2.

In the following, we will focus on the semantics of the OR-join connector. Its main characteristics is the delay of the propagation of a process folder on one of its incoming arcs as long as a process folder could possibly arrive at one of the other incoming arcs. Thus, the semantics of the OR-join is *non-local*[1]. In order to decide whether a process folder at an OR-join can be immediately propagated or whether the propagation should be delayed, we need to know whether there will be a process folder arriving at one of the other arcs at some time in the future. This information is not local to the OR-join, but may depend on all other parts of the EPC. In our example, it depends on the choice taken at the XOR-split in the left-hand part of the EPC.

3 A technical problem

In principle, there is no problem with the non-locality of the informal semantics of EPCs. But, a formalization requires some care, because the definition of such a semantics in some way refers to itself again: The semantics to be defined (propagation of process folders) occurs in its own definition (propagation of process folders to an incoming arc of an OR-join). This may easily result in a cyclic definition without any meaning. And this is what happened in the definition of the semantics in [Rum99, NR02]: There, a state is the assignment of process folders to the different nodes of the EPCs, and a *transition relation* $\rightarrow_{TR}$ between these states defines the state changes. Unfortunately, the transition relation $\rightarrow_{TR}$ occurs in its own definition again.

One way out of this cyclic definition would be to consider the definition as a fixed-point equation. Then, $\rightarrow_{TR}$ would be the least fixed-point of this equation. Unfortunately, the relation $\rightarrow_{TR}$ occurs under a negation in its definition, to the effect that the fixed-point does not exist in all cases[2]. Fortunately, fixed-points seem to exist in all practically relevant cases. In particular, a fixed-point exists for the example given in the next section.

[1]Here, we focus on the non-locality of the OR-join connector. According to Nüttgens and Rump [NR02], the XOR-join connector has a non-local semantics, too. But we do not deal with the XOR-join in the rest of the paper.

[2]For the experts, the Appendix shows an example of an EPC, for which a recursive interpretation of the definition of $\rightarrow_{TR}$ runs into an infinite cycle.

4 A conceptual problem

Figure 2 shows another EPC[3] with two OR-joins in a feedback loop, which is a vicious circle, as we will see. With the above mentioned fixed-point interpretation, the semantics of [NR02] is that the process folders are stuck at f1 and f2. The two OR-joins will not propagate the process folders to the Inner events.

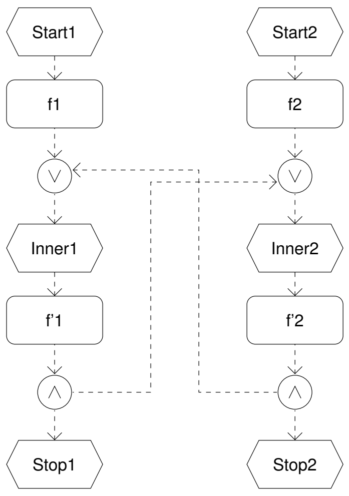

Figure 2: A vicious circle

Is this the intended semantics of this EPC? We will argue that it is not. To this end, we consider the OR-join above the Inner1 event. Since the Inner2 event will never occur, we know that no process folder will ever arrive at the other incoming arc of the OR-join. So, according to the informal semantics, the OR-join should propagate the process folder from f1 to the event Inner1. Symmetrically, we can argue that the process folder from f2 should be propagated to Inner2. So, we have shown that the process folders should not be delayed at f1 and f2 according to the informal semantics of EPCs.

Is this the intended semantics of this EPCs? Again, we will argue that it is not. We will argue that the OR-joins should not propagate the process folders from f1 and f2. To this end, we consider the OR-join before the Inner1 event again. Since Inner2 will eventually occur, we know that eventually there will be a process folder arriving at the second incoming arc. According to the informal semantics, this implies that the OR-join should wait with the propagation of the process folder until the second folder arrives. Symmetrically, we can argue that the process folder from f2 should not be propagated. So, we know that the process folders should be delayed at f1 and f2 according to the informal semantics of EPCs.

[3]Rump [Rum99] gives a similar example. But his point is that, in some situations, OR-joins may result in a deadlock. Here, we argue that the situation is much worse: the intuitive semantics of EPCs fails.

Altogether, we have established a vicious circle. In a situation with process folders on f1 and f2, we do not know whether the OR-joins should propagate the process folders or not. Both alternatives result in a contradiction to the informal semantics of EPCs. This shows that there is no formal semantics that precisely captures the informal semantics of EPCs.

Every formalization will deviate from the informal semantics in some way or the other. Either it will impose some syntactical restrictions on EPCs, or it will deviate from the informal semantics in some way. We believe that this choice depends on the purpose of the semantics.

5 A practical problem

In the previous sections, we have shown that the non-locality of the semantics of EPCs results in some technical and conceptual problems. But, non-locality is problematic from a practical point of view, too. The reason is that, for routing the process folders through an EPC, one needs to know the complete EPC, since process folders could arrive at an OR-join from far away parts of the EPC. This is acceptable for workflows of a single organization[4]. For inter-organizational workflows, however, this is not acceptable anymore, because different parts of the workflow belong to different organizations, which will not reveal their part to the others. Due to the non-locality of the semantics of EPCs, this would be necessary. Another solution to this problem would be a mechanism for a check-back with the other organizations whether a process folder will be arriving from them in the future. This does not not appear to be a realistic situation. Even worse, there could be a vicious cycle of check-backs.

6 Pragmatic solutions

Though the non-locality of the OR-join has its complications, it is a pattern frequently occurring in a wide variety of workflow processes. Therefore, dealing with this problem is a practical issue, and vendors of workflow management systems have found ways to deal with it. Most of the systems require the workflow designer to replace OR-splits and OR-joins by AND/XOR-splits and AND/XOR-joins. However, there are also systems that directly offer constructs that resemble the OR-join as discussed in this paper. For an overview of these systems we refer to [AHKB02, Kie02] where 15 systems are evaluated on the basis of 20 workflow patterns. Pattern 7, also referred to as the *synchronizing merge*, corresponds to the OR-join with a non-local semantics. The synchronization merge is fully or partially supported by InConcert, Eastman, Domino Workflow, eProcess, and MQSeries Workflow. In these systems, the designer does not have to specify the type of the join; this is automatically handled by the system. Here, we will briefly discuss how these systems support the synchronizing merge, i. e. the OR-join with a non-local semantics.

[4]Actually, it is only acceptable if the routing is performed by a workflow management system. Nobody wants to calculate the complete semantics of an EPC by hand just for routing the process folders.

InConcert (TIBCO, [Tib00]) only allows for acyclic workflow graphs and does not allow for XOR-splits. The only way to model alternatives is through adding conditions to tasks, i. e., a task is simply skipped when the condition evaluates to *false*. This way the problem of the OR-join is avoided and every join becomes an AND-join. This solution seems to be very limiting but is motivated by the fact that InConcert allows for ad-hoc workflows. End-users can make changes on-the-fly. Therefore, the language has to be very simple and it should be impossible to design a workflow with deadlocks or livelocks.

Eastman (Eastman, [Sof98]) directly supports the synchronizing merge in most cases. There are no syntactical requirements like the absence of cycles. However, in certain circumstances the OR-join does not behave as expected. It appears that, in the system, there are some heuristics to determine whether to wait or not. In most practical cases these heuristics work. However, for more involved situations the heuristics fail in the sense that the OR-join progresses while there is still some input on its way.

Domino Workflow (Lotus/IBM, [NEG$^+$00]) also supports the synchronizing merge without imposing syntactical requirements. It is not clear how the engine detects whether it should wait or not. For all practical examples, the solution seems to work satisfactory. However, Domino Workflow has not been tested using nasty examples like the “vicious circle”.

eProcess (FileNET, [Fil02]) supports the synchronizing merge, but requires a one-to-one correspondence between AND/OR-splits and AND/OR-joins, and every path starting in an AND/OR-split should lead to the corresponding AND/OR-join. Then, the OR-join can be implemented by a counter which is set by the split and inspected and decreased by the join. After executing the split, the counter is set to the number of parallel threads[5] initiated. Then, the counter is decreased by one for every thread reaching the join. The moment the counter is set to zero, processing continues. It is interesting to see that the simple syntactical restriction to a one-to-one correspondence between splits and joins makes the semantics of the OR-join local.

MQSeries Workflow (IBM, [IBM99]) is one of the most interesting systems when it comes to the OR-join. MQSeries Workflow only allows for the iteration of entire subprocesses, i. e., cycles are not allowed in the workflow graph. Note that this requirement does not imply that there is one-to-one correspondence between splits and joins (like for eProcess). Nevertheless this requirement is sufficient for implementing the synchronizing merge using a local rule. MQSeries workflow uses a technique called “dead-path elimination” [LR99, IBM99]: Initially, each input arc is in state “unevaluated”. As long as one of the input arcs is in this state, the activity is not enabled. The state of an input arc is changed to true the moment the preceding activity is executed. However, to avoid deadlocks, the input arc is set to false the moment it becomes clear that it will not fire. By propagating these false signals, deadlocks are excluded and the resulting semantics is that of a synchronizing merge. The solution used in MQSeries workflow is similar to having true and false tokens in Petri nets: true tokens should be executed while false tokens are simply passed on. The idea of having true and false tokens to address complex synchronizations was already reported in [GT84]. However, the bipolar synchronization schemes presented in

[5]In the semantics of EPCs, the threads could be considered as process folders.

[GT84] are primarily aimed at avoiding constructs such as the synchronizing merge, i. e., the nodes are pure AND/XOR-splits/joins and partial synchronization is neither supported nor investigated[6].

The examples listed above show that the issue of non-local OR-joins is addressed by contemporary workflow management systems. Systems like InConcert, eProcess, and MQSeries have solved the OR-join problem using syntactical restrictions. Other systems like Eastman and Domino Workflow seem to use a non-local semantics similar to the one discussed in this paper. Such a non-local semantics may lead to unexpected results. Moreover, a non-local semantics may result in poor performance as is stated in the manual of Eastman: "Parallel instances can accumulate at a Join workstep if the instances are routed to the workstep by preprocessing rules. These instances will eventually be joined by a RouteEngine subprocess (thread) that examines Join worksteps for such instances. This *Join scavenger thread* reduces system efficiency, so routing to Join worksteps using preprocessing rules should be avoided" [Sof98].

7 Conclusion

In this paper, we have discussed a problem of the EPC semantics of Nüttgens and Rump [Rum99, NR02]. Actually, the problem is inherent to the informal semantics of EPCs. It came into focus thanks to Nüttgen's and Rump's attempt to precisely capture the informal semantics. We have shown that this is impossible, and that a fully compliant formal semantics does not exist.

Maybe, there is a sound formal semantics that captures the most important aspects of non-locality of the informal semantics. We see several possibilities to achieve this goal. One way would be the above mentioned fixed-point interpretation; but this would require to syntactically exclude those EPCs for which the fixed-point does not exist. Another way would be to define two transition relations; the first would be completely local, the second would be non-local, but would only use the first for checking whether a folder can arrive at an OR-join. But, all these semantics will be quite complex and will have pitfalls, to the effect that the semantics neither will help to understand EPCs, nor will help to develop analysis methods, nor will provide efficient routing algorithms.

There are several proposals of much simpler semantics for EPCs, which deliberately have chosen to be local – either by excluding the problematic connectors (i. e. the OR-join) or by giving them a local semantics. Clearly, they do not fully capture the informal semantics, but there simplicity is an argument in favour of these semantics.

Before starting another effort in defining the semantics of EPCs, we think that a discussion of the degree of non-locality that a semantics for EPCs should and could have is in order. For this discussion, we need a set of well-accepted practical EPC models of business processes, which cover the typical situations in which OR-joins and XOR-joins occur, along with the intended behaviour.

[6]Note that Langner et al. [LSW98] build on the work in [GT84] to analyze EPCs.

References

[AHKB02] W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow Patterns. QUT Technical report, FIT-TR-2002-02, Queensland University of Technology, Brisbane, 2002. (Accepted for publication in Distributed and Parallel Databases, also see http://www.tm.tue.nl/it/research/patterns.).

[Fil02] FileNET. *Panagon eProcess Designer 4.2.2*. FileNET Corporation, Costa Mesa, CA, USA, 2002.

[GT84] H. J. Genrich and P. S. Thiagarajan. A Theory of Bipolar Synchronization Schemes. *Theoretical Computer Science*, 30(3):241–318, 1984.

[IBM99] IBM. *IBM MQSeries Workflow - Getting Started With Buildtime*. IBM Deutschland Entwicklung GmbH, Boeblingen, Germany, 1999.

[Kie02] B. Kiepuszewski. *Expressiveness and Suitability of Languages for Control Flow Modelling in Workflows (submitted)*. PhD thesis, Queensland University of Technology, Brisbane, Australia, 2002.

[KNS92] G. Keller, M. Nüttgens, and A.-W. Scheer. Semantische Prozessmodellierung auf der Grundlage Ereignisgesteuerter Prozessketten (EPK). Veröffentlichungen des Instituts für Wirtschaftsinformatik (IWi), Heft 89, Universität des Saarlandes, January 1992.

[LR99] F. Leymann and D. Roller. *Production Workflow: Concepts and Techniques*. Prentice-Hall PTR, 1999.

[LSW98] P. Langner, C. Schneider, and J. Wehler. Petri Net Based Certification of Event driven Process Chains. In J. Desel and M. Silva, editors, *Application and Theory of Petri Nets 1998*, *LNCS* 1420, pages 286–305. Springer, 1998.

[NEG$^+$00] S.P. Nielsen, C. Easthope, P. Gosselink, K. Gutsze, and J. Roele. *Using Lotus Domino Workflow 2.0, Redbook SG24-5963-00*. IBM, Poughkeepsie, USA, 2000.

[NR02] Markus Nüttgens and Frank J. Rump. Syntax und Semantik Ereignisgesteuerter Prozessketten (EPK). In *PROMISE 2002, Prozessorientierte Methoden und Werkzeuge fürr die Entwicklung von Informationssystemen, GI Lecture Notes in Informatics* P-21, pages 64–77. Gesellschaft für Informatik, 2002.

[Rum99] Frank J. Rump. *Geschäftsprozeßmanagement auf der Basis ereignisgesteuerter Prozeßketten*. Teubner-Reihe Wirtschaftsinformatik. B.G.Teubner, 1999.

[Sof98] Eastman Software. *RouteBuilder Tool User's Guide*. Eastman Software, Inc, Billerica, MA, USA, 1998.

[Tib00] Tibco. *TIB/InConcert Process Designer User's Guide*. Tibco Software Inc., Palo Alto, CA, USA, 2000.

Acknowledgment We would like to thank Matthias Gehrke for his comments on an earlier version of this paper.

Appendix

Figure 3 shows a nasty example of an EPC. For this EPC, a recursive interpretation of the definition of $\rightarrow_{TR}$ in [NR02] runs into an infinite cycle, when started with a process folder on Start1 and Start2.

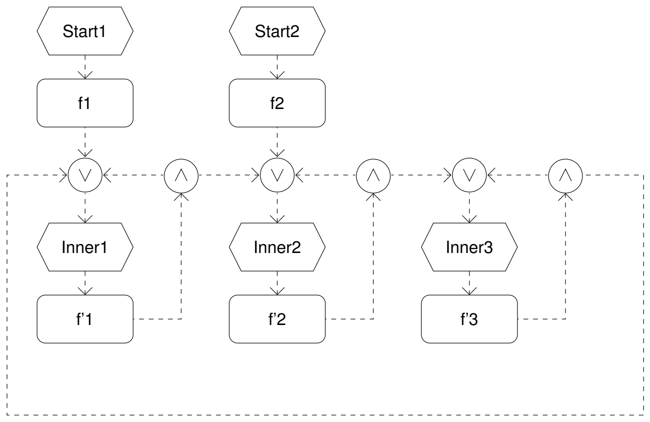

Figure 3: A vicious circle worse than the one from Fig. 2