Business Processes Modelling

MPB (6 cfu, 295AA)

Roberto Bruni
http://www.di.unipi.it/~bruni

20 - EPC analysis
We overview the main challenges that arise when analysing EPC diagrams with Petri nets.

Ch. 6 of Business Process Management: Concepts, Languages, Architectures
EPC Diagrams
EPC ingredients at a glance

Event

Function

Connectors

Control Flow
EPC: Example
EPC Semantics
We exploit the formal semantics of nets to give unambiguous semantics to EPC diagrams.

We transform EPC diagrams to Workflow nets: the EPC diagram is sound if its net is so.

We can reuse the verification tools to check if the net is sound.

Is there a unique way to proceed? Not necessarily!
Translation of EPC to Petri nets
The idea

From EPC to wf nets in three steps

1. **Step 1**
   - Convert each event, function, or connector to a net fragment.

2. **Step 2**
   - Connect fragments together.

3. **Step 3**
   - Enforce initial and final place.
Step 1

We replace each event, function and connector separately with small net fragments
Step 2: dummy style

Then we connect the fragments together (we may decide to introduce *dummy places / transitions*)
Step 2: fusion style

Then we connect the fragments together (or we may decide to merge places / transitions)
Step 3: unique start

The paths finally join. The EPC syntax rules state that: For every two elements there is a path between their corresponding events in the EPC.

Applying Step 1 and Step 2, an EPC is translated into a Petri net but not necessarily into a WF-net. If the EPC contained more than one start, and/or end event, the resulting net may have more than one start and/or sink place. Therefore, a new start place and/or a new sink place is added.

These new places are connected to the Petri net so that the places representing the primary start events (or end events) of the EPC are initialized (cleaned up). The connection of the new places to the primary places is not trivial and depends on the syntax-rules that restrict the number of start and end events. Moreover, if there are several start events (or end events), it is not clear whether they are mutually exclusive or parallel.

3.3.3. Step 3: Adding unique input/output places

Figure 4: Transformation of the OR-connector.
Step 3: unique end

The paths finally join. The EPC syntax rules state that: For every two elements there is a path between different start events (end events), until they join

Applying Step 1 and Step 2, an EPC is translated into a Petri net but not necessarily into a unique Petri net. The connection of the new places to the primary places is not trivial and depends on the primary start events (or end events) of the EPC are initialized (cleaned up). The resulting net may have more than one start and/or sink place. There are no EPC syntax-rules that restrict the number of start and end events. Moreover, if there are several start events (or end events), it is not clear whether they are mutually exclusive or parallel. Therefore, a new start place and/or a new sink place is added.

OR end

(sometimes XOR/AND can be preferred)
Three approaches

We overview three different translations

<table>
<thead>
<tr>
<th>n.</th>
<th>ingenuity</th>
<th>style</th>
<th>applicability</th>
<th>outcome</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st</td>
<td>easy</td>
<td>fusion</td>
<td>any EPC</td>
<td>likely unsound, (relaxed soundness)</td>
</tr>
<tr>
<td>2nd</td>
<td>medium, context dependent</td>
<td>(dummy)</td>
<td>simplified EPC: event function alternation, no OR connectors</td>
<td>free-choice net</td>
</tr>
<tr>
<td>3rd</td>
<td>hard, context dependent</td>
<td>dummy</td>
<td>decorated EPC: join-split correspondence, OR policies</td>
<td>accurate analysis</td>
</tr>
</tbody>
</table>
Commonalities

EPC element

- **A** (event)
- **f** (function)

Control flow

Net fragment

- **place**
- **transition**

Arc
First attempt
(straight translation)

Relaxed Soundness of Business Processes

Juliane Dehnert¹,* and Peter Rittgen²

¹ Institute of Computer Information Systems, Technical University Berlin, Germany
dehnert@cs.tu-berlin.de

² Institute of Business Informatics, University Koblenz-Landau, Germany
rittgen@uni-koblenz.de
Rationale

EPC success is due to its **simplicity**

EPC diagrams lack a consistent semantics: **ambiguous and flawed** process descriptions can arise in the **design phase**

It is important to find out **flaws** as **soon** as possible

Therefore

We need to fix a **formal representation** that **preserves all ambiguities**
Step 1: AND split

EPC element

Net fragment
Step 1: AND join

EPC element

net fragment
Step 1: XOR split

EPC element

> net fragment
Step 1: XOR join

EPC element

net fragment
Step 1: OR split

EPC element

net fragment

 xor
+ and
Step 1: OR join

EPC element

net fragment
Step 2: fusion style

Element fusion (case 1)

Arc fusion (case 2)

Applying Step 1 and Step 2, an EPC is translated into a Petri net but not necessarily into a WF-net. If the EPC contained more than one start, and/or end event, the resulting net may have more than one start and/or sink place. There are no EPC syntax rules that restrict the number of start and end events. Moreover, if there are several start events (or end events), it is not clear whether they are mutually exclusive or parallel. Therefore, a new start place and/or a new sink place is added. These new places are connected to the Petri net so that the places representing the primary start events (or end events) of the EPC are initialized (cleaned up). The connection of the new places to the primary places is not trivial and depends on the relation of the corresponding events in the EPC.

One way to determine the relation would be to track the paths, starting from the different start events (end events), until they join. The paths finally join. The EPC syntax rules state that: For every two elements there is a path.
Example

Sound?
Example

Step 1
events and functions
Example

Step 1 connectors

AND split

XOR split

XOR join

AND split
Example

Step 2 fusion
Example

Step 2 fusion
Example

Step 3
unique end

implicit AND join (because of A2)
Example

Step 3
unique end

implicit AND join (because of A2)
Example

Sound?

Steps 1+2+3
Soundness analysis

Not sound!
Soundness analysis
Soundness analysis
Soundness analysis

the right thing to do would be to fire O1e
Soundness analysis

the right thing to do would be to fire O1e
Soundness analysis

but O1f and O1d are enabled as well (OR semantics!)
Soundness analysis

proper completion is not guaranteed
(N* unbounded)
Soundness analysis

proper completion is not guaranteed (N* unbounded)
Soundness analysis

Can we repair the model?
Soundness analysis

AND join instead of OR join?
Soundness analysis

Not sound!

Semantical analysis

- Qualitative analysis
- Structural analysis
  - Net statistics
    - Wrongly used operators: 0
    - Free-choice violations: 0
  - S-Components
    - S-Components: 2
    - Places not covered by S-Comp
      - O1a
  - Wellstructuredness
    - PT-Handles: 2
    - TP-Handles: 1
  - Soundness
    - Workflow net property
    - Initial marking
    - Boundedness
    - Liveness
      - Dead transitions: 0
      - Non-live transitions: 12
Soundness analysis
Soundness analysis
Soundness analysis

the right thing to do would be to fire X1b

AND join instead of OR join?
Soundness analysis

the right thing to do would be to fire X1b

AND join instead of OR join?
Soundness analysis

but X1a is enabled as well

AND join instead of OR join?
Soundness analysis

possible deadlock! option to complete is not guaranteed (N* non-live)

AND join instead of OR join?
Soundness analysis

AND join instead of OR join + ad hoc flow?

we miss a token in O1a
Soundness analysis

AND join instead of OR join + ad hoc flow?
Soundness analysis

Sound!
Soundness analysis

Sound, but…
we have repaired the wf net, not the original EPC diagram!
Soundness analysis
Soundness analysis

The diagram is now more complex and less readable than the original one!

Are we sure that its translation is the same sound wf net that we have designed ad hoc?

Are we sure it is sound?

Need to restart the analysis!!
Relaxed Soundness (optional reading)
Problem

EPC is widely adopted
also at early stages of design

WF nets offer a useful tool

but

Soundness can be too demanding at early stages
(Un)sound behaviours

A **sound** behaviour:
we move from a start event to an end event
so that nothing blocks or remains undone

The language of the net
collects all and only
its sound behaviours

\[ L(N) = \{ \sigma \mid i \xrightarrow{\sigma} o \} \]

Execution paths leading to **unsound** behaviours
can be used to infer potential mistakes
Relaxed soundness

If some unsound behaviour is possible but any transition can take part to one sound execution, then the process is called **relaxed sound**

**Definition:** A WF net is **relaxed sound** if every transition belongs to a firing sequence that starts in state $i$ and ends in state $o$ (i.e. it appears in the language of the net)

$$\forall t \in T. \exists \sigma \in L(N). \bar{\sigma}(t) > 0$$
Example

Relaxed sound?

Steps $1+2+3$
Example

Relaxed sound?

Steps 1+2+3

da sound execution
Example

Relaxed sound?

Steps 1+2+3
Example

Relaxed sound?

Steps 1+2+3
Example

Not relaxed sound as a net!

Steps 1+2+3

one task not involved in some sound execution
Example

Relaxed sound as EPC!

Steps 1+2+3

all EPC nodes involved in some sound execution
Relaxed soundness?

If the WF net is not relaxed sound there are transitions that are not involved in sound executions (not included in a firing sequence of L(N))

Their EPC counterparts may need improvements

Relaxed soundness can be proven only by enumeration (of enough firing sequences of L(N))

Open problem
No equivalent characterization is known that is more convenient to check
Abstract
For many companies, business processes have become the focal point of attention. As a result, many tools have been developed for business process engineering and the actual deployment of business processes. Typical examples of these tools are BPR (Business Process Reengineering) tools, ERP (Enterprise Resource Planning) systems, and WFM (Workflow Management) systems. Some of the leading products, e.g. SAP R/3 (ERP/WFM) and ARIS (BPR), use Event-driven Process Chains (EPCs) to model business processes. Although event-driven process chains have become a widespread process modeling technique, they suffer from a serious drawback: neither the syntax nor the semantics of an event-driven process chain are well defined. In this paper, this problem is tackled by mapping event-driven process chains (without connectors of type OR) onto Petri nets. Petri nets have formal semantics and provide an abundance of analysis techniques. As a result, the approach presented in this paper gives formal semantics to event-driven process chains. Moreover, many analysis techniques become available for event-driven process chains. To illustrate the approach, it is shown that the correctness of an event-driven process chain can be checked in polynomial time by using Petri-net-based analysis techniques.

Keywords:
Event-driven process chains, Petri nets, workflow management, verification.

1 Introduction
As a response to increased competitive pressure in the global marketplace, enterprises are looking to improve the way they are running their businesses. The term business process engineering subsumes the set of principles, activities, and
Simplified EPC

We restrict the analysis to a sub-class of EPC diagrams

We require:

- **event / function alternation**
  (also along paths between two connectors)
  (fusion not needed, dummy places/transitions not needed)

- **OR-connectors are not present**
  (avoid intrinsic problems with OR join)
OR-connectors are not present
alternation is not satisfied

Example

Add dummy events and functions to force alternation

Step 0
Example

Step 1
events and functions
Step 1: split/join connectors

The translation of logical connectors depends on the context:

if a connector connects **functions to events**
we apply a certain translation

if it connects **events to functions**
we apply a different translation
Step 1: split/join connectors

The translation of logical connectors depends on the context:

if a connector connects transitions to places we apply a certain translation

if it connects places to transitions we apply a different translation
Step 1: AND split

EPC

(event to functions)

(functions to events)
Step 1: AND join

(event to functions)
Example

Step 1
AND connectors
Step 1: XOR split

/event to functions/

EPC

\[ \text{XOR} \quad \rightarrow \]

\[ e_1 \quad \rightarrow \quad f_1 \quad \rightarrow \quad f_2 \]

\[ \text{(event to functions)} \]

\[ \text{net fragment} \]

\[ e_1 \quad \rightarrow \quad f_1 \quad \rightarrow \quad f_2 \]

\[ \text{functions to events} \]

EPC

\[ \text{XOR} \quad \rightarrow \]

\[ f_1 \quad \rightarrow \quad e_1 \quad \rightarrow \quad e_2 \]

\[ \text{net fragment} \]

\[ f_1 \quad \rightarrow \quad e_1 \quad \rightarrow \quad e_2 \]
Step 1: XOR join

(event to functions)

(functions to events)
Example

Step 1: XOR connectors
Overall strategy

From any EPC we derive a free-choice net
Example

Sound?
Example

Sound?

Steps 1+2(+3)
Example

Not sound!
Third attempt
(decorated EPC)

PETER RITTGEN

MODIFIED EPCs AND THEIR FORMAL SEMANTICS
Decorated EPC

Applicable to any EPC diagram, provided that its designer add some information

We require:

every (X)OR join is paired with a corresponding split
(possibly of the same type)

OR-joins are decorated with a policy
(avoid OR join ambiguous behaviour)
Step 1: AND split

EPC element

∧

net fragment
Step 1: XOR split

EPC element

```
XOR
```

→

net fragment

```

```
Step 1: OR split

EPC element

net fragment

xor
+ 
and

88
Step 1: AND join

EPC element

net fragment
XOR join: intended meaning

if both inputs arrive, it should block the flow

if one input arrives, it cannot proceed unless it is informed that the other input will never arrive
OR join: intended meaning

if only one input arrives,
it should release the flow

if both inputs arrive,
it should release only one output

if one input arrives,
it must wait until the other arrives or
it is guaranteed that the other will never arrive
OR join: assumption

If an OR join has a matching split, its semantics is wait-for-all: wait for the completion of all activated paths.

Otherwise, also other policies can be chosen:

- **every-time**: trigger the outgoing path on each input
- **first-come**: wait for the first input and ignore the second

**Assumption**: every OR join is tagged with a policy (some suggested to have different trapezoid symbols)
Example

two OR joins but no OR split
Example

only one candidate split
Example

two candidate splits
Example

assign corresponding splits
Example

assign policies

wfa

fc
Assumption

...  

An OR join with matching split uses wfa

If an OR join has non-matching corresponding split it is decorated with a policy (wfa, fc, et)

wfa: wait-for-all  
works well with any corresponding split

...
Step 1: OR join (wfa)

EPC element

net fragment

matching split

wfa
Step 1: OR join (wfa)
Step 1: OR join (wfa)

EPC element

net fragment

EPC element

Step 1: OR join (wfa)
Assumption

... 

If an OR join has non-matching corresponding split it is decorated with a policy (wfa, fc, et)

et: every-time works well with corresponding XOR split

...
Step 1: OR join (et)

corresponding XOR split

EPC element

net fragment
Step 1: OR join (et)

EPC element

corresponding AND split

net fragment

\( s \) (s) 

\( j \) (et)

every time: any token gets through (multiple tokens may appear in the target)
Assumption

If an OR join has non-matching corresponding split it is decorated with a policy (wfa, fc, et)

fc: first-come
works well with corresponding XOR split

...
Step 1: OR join (fc)

EPC element

s

XOR

Corresponding XOR split

j (s)

fc

net fragment
Step 1: OR join (fc)

EPC element

\[ \square \land (s) \]  

net fragment

\[ \Rightarrow \]

first come: at most one token gets through
(pending tokens may remain)
**XOR join: assumption**

If a XOR join has a **matching split**, the semantics is:

“it blocks if both paths are activated and it is triggered by a unique activated path”

Any policy (wait-for-all, first-come, every-time) **contradicts the exclusivity** of XOR

(a token from one path can be accepted only if we make sure that no second token will arrive via the other path)

**Assumption:** every XOR join has a matching split

(the implicit start split is allowed as a valid match)
Assumption

... Any XOR join has a corresponding matching split ...
Step 1: XOR join

EPC element

net fragment

matching split

j (s)
Step 2: dummy style

straight conversion

straight conversion
Step 2: dummy style

- Circle needs a dummy transition
- Circle needs a dummy transition
- Square needs a dummy place
Example

fc  Sound?
Example

Step 1 events and functions
Example

Step 1 splits
Example

Step 1
splits and
joins

wfa

fc
Example

Step 2(+3) dummy style

wfa
Example

Sound?  
Steps 1+2(+3)

wfa  
fc
Example

Not sound!
EPC pros and cons

You may leave complete freedom, but most diagrams will not be sound

You may constrain diagrams, but people like flexible syntax and ignore guidelines

You may require to add decorations, but people will be lazy or misinterpret policies
Exercise

Is this EPC diagram sound? Choose one of the three techniques seen and apply it to answer the above question.