Business Processes Modelling

MPB (6 cfu, 295AA)

Roberto Bruni

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

21 - Event-driven process chains
We overview EPC and the main challenges that arise when analysing them with Petri nets.
EPC origin (early 1990's)

EPC method originally developed as part of a holistic modelling approach called ARIS framework (Architecture of Integrated Information Systems) by Wilhelm-August Scheer
Event-driven Process Chain

An Event-driven Process Chain (EPC) is a flow-chart that can be used:
to configure an Enterprise Resource Planning implementation
to drive the modelling, analysis, redesign of business process

Informal notation: simple, intuitive and easy-to-understand

EPC represents domain concepts and processes
(neither their formal aspects nor their technical realization)

EPC Markup Language (EPML): XML interchange format
EPC Diagrams
Why do we need diagrams?

Graphical languages communicate concepts

Careful selection of symbols
shapes, colors, arrows
(the alphabet is necessary for communication)

Greatest common denominator of the people involved

Intuitive meaning
(verbal description, no math involved)
An EPC is a graph of events and functions. It provides some logical connectors that allow alternative and parallel execution of processes (AND, XOR, OR).
EPC ingredients at a glance
Events

Any EPC diagram must start / end with event(s)

Graphical representation: hexagons

Passive elements used to describe under which circumstances a process (or a function) works or which state a process (or a function) results in (like pre- / post-conditions)
Functions

Any EPC diagram may involve several **functions**

Graphical representation: rounded rectangles

Active elements used to describe the tasks or activities of a business process

Functions can be refined to other EPC diagrams
Logical connectors

Any EPC diagram may involve several connectors

Graphical representation: circles (or also octagons)

\[
\begin{align*}
\text{AND} & \quad \bigwedge \\
\text{XOR} & \quad \times \\
\text{OR} & \quad \bigvee
\end{align*}
\]

Elements used to describe
the logical relationships between split/join branches
Control flow

Any EPC diagram may involve several connections

Graphical representation: dashed arrows

Control flow is used to connect events with functions and connectors by expressing causal dependencies
EPC diagrams

EPC elements can be combined in a fairly free manner (possibly including cycles)

The graph is **weakly connected** (e.g., no isolated nodes)

**Events** have at most one incoming and one outgoing arc
Events have at least one incident arc
There must be at least one start event and one end event

**Functions** have exactly one incoming and one outgoing arc

**Connectors** have either one incoming arc and multiple outgoing arcs or viceversa (multiple incoming arcs and one outgoing arc)
Logical connectors: splits and joins

Splits

Joins
EPC: Example

\[ \wedge = \text{AND} \]

\[ X = \text{XOR} \]

\[ \vee = \text{OR} \]
EPC Diagrams: guidelines

Other constraints are sometimes imposed

Unique start / end event

No direct flow between two events
No direct flow between two functions

No event is followed by a decision node (i.e. (X)OR-split)
EPC guidelines: Example

direct flow between functions

multiple end events
Problem with guidelines

From empirical studies:
guidelines are too restrictive and people ignore them
(otherwise diagrams would get unnecessarily complicated,
more difficult to read and understand)

Solution:
It is safe to drop most constraints
(implicit dummy nodes might always be added later, if needed)
EPC: repairing alternation

add dummy functions to guarantee alternation
EPC: repairing alternation

add dummy events to guarantee alternation
EPC: repairing decisions

add dummy nodes to guarantee no event be followed by a decision node ((X)OR-split)
EPC: repairing multiple start events

A start event is an event with no incoming arc. It invokes a new instance of the process template.

Start events are mutually exclusive.

Assume an implicit XOR split is present.
EPC: repairing multiple end events

An end event is an event with no outgoing arc. It indicates completion of some activities. What if multiple end events occur? No unanimity! They are followed by an implicit join connector (typically a XOR… but not necessarily so).

Assume an implicit join is present.
Other ingredients: function annotations

**Organization unit:**
determines the person or organization responsible for a specific function
(ellipses with a vertical line)

**Information, material, resource object:**
represents objects in the real world
e.g. input data or output data for a function
(rectangles linked to function boxes)
angles with vertical lines on its sides)

**Supporting system:** technical support
(rectangles with vertical lines on its sides)
EPC Semantics
EPC intuitive semantics

A process starts when some initial event(s) occurs

The activities are executed according to the constraints in the diagram

When the process is finished, only final events have not been dealt with

If this is always the case, then the EPC is “correct”
EPC formal semantics?

Little unanimity around the EPC semantics

Rough verbal description in the original publication by Scheer (1992)

Later, several attempts to define formal semantics (assigning different meanings to the same EPC, sometimes leading to paradoxes)

Discrepancies typically stem from the interpretation of (X)OR join connectors
Sound EPC diagrams

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

Step 1: Convert each
- event
- function
- connector
  to a net fragment

Step 2: Connect fragments together

Step 3: Enforce initial place
    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

Start1  Start2

Steps 1+2

Start1  Start2

Step 3
unique start

XOR start

start1  start2
Step 3: unique end

OR end
(sometimes XOR/AND can be preferred)
Three approaches

We overview three different translations

<table>
<thead>
<tr>
<th>n.</th>
<th>trickiness</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</td>
<td>(dummy)</td>
<td>simplified EPC</td>
<td>free-choice net</td>
</tr>
<tr>
<td></td>
<td>dependent</td>
<td></td>
<td>event function alternation, no OR connectors</td>
<td></td>
</tr>
<tr>
<td>3rd</td>
<td>hard, context</td>
<td>dummy</td>
<td>decorated EPC</td>
<td>accurate analysis</td>
</tr>
<tr>
<td></td>
<td>dependent</td>
<td></td>
<td>join-split correspondence, OR policies</td>
<td></td>
</tr>
</tbody>
</table>
Commonalities

EPC element

event

function

place

transition

control flow

arc

net fragment
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

Figure 4: Transformation of the OR-connector.

3.3.3. Step 3: Adding unique input/output places

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.

Figure 5: Step 3: Adding new start and sink places.

One way to determine the relation would be to track the paths, starting from the different start events (end events), until they join. The connection of the new place to the Petri net is not straightforward. 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

OR join
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)
EPC

Example

Sound?

Steps 1+2+3

goods arrived

AND

check goods

XOR

ok

not ok

complaint

data revised

XOR

AND

store goods

record receipts of goods

stored

goods recorded

goods arrived

AND

goods arrived

A1a

A1

A1b

check goods

X1a

X1

X1b

ok

not ok

complaint

data revised

X2a

X2

X2b

A2a

A2

A2b

O1a

O1

O1e

O1f

store goods

stored

A3

A1\Rightarrow A1b

A2\Rightarrow A2b

O1\Rightarrow O1b
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!
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

AND join instead of OR join?

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

Semantical analysis

- Qualitative analysis
- Structural analysis
  - Net statistics
    - Wrongly used operators: 0
    - Free-choice violations: 0
    - S-Components
  - Wellstructuredness
    - PT-Handles: 2
    - TP-Handles: 4
  - Soundness
    - Workflow net property
    - Initial marking
    - Boundedness
    - Liveness

Sound!
Soundness analysis

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

[Diagram showing a process flow with nodes and edges representing actions such as 'check goods', 'data revised', 'record receipts of goods', 'store goods', 'end', etc.]

[Diagram on the right is a possible variation or an alternative process flow showing different outcomes and decisions points such as 'not ok', 'complaint', 'ok', 'data revised', 'record receipts of goods', 'stored', 'goods recorded', etc.]

Question mark indicating uncertainty or comparison between the two diagrams.
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)

\[ \forall t \in T. \exists \sigma \in L(N). \tilde{\sigma}(t) > 0 \]
Example

Relaxed sound?

Steps 1+2+3
Example

Relaxed sound?

Steps 1+2+3

a sound execution
Example

Relaxed sound?

Steps 1+2+3

another sound execution
Example

Relaxed sound?

Steps 1+2+3

tasks involved in some sound execution
One task not involved in some sound execution

Example

Steps 1+2+3

Not relaxed sound as a net!
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
Formalization and Verification of Event-driven Process Chains

W.M.P. van der Aalst
Department of Mathematics and Computing Science, Eindhoven University of Technology, P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands, telephone: +31 40 2474295, e-mail: wsinwa@win.tue.nl
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

(event to functions)

(functions to events)
Step 1: AND join

(event to functions)

(functions to events)
Example

Step 1
AND connectors
Step 1: XOR split

EPC

net fragment

(event to functions)

EPC

net fragment

(functions to events)
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

net fragment
Step 1: OR split

EPC element

net fragment

xor
+ and
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
A **candidate split** for a join node is any split node whose outputs are connected to the inputs of the join.

- **s1** is a candidate split for **j1**
- **s1** and **s2** are candidate splits for **j2**
Corresponding split

A *corresponding split* for a join node is a chosen candidate split.

- **s1** as a corresponding split for **j1**
- **s2** as a corresponding split for **j2**

We tag each join with its corresponding split.
Matching split

A corresponding split for a join node is called **matching** if it has the same type as the join node.

- $s_1$ is a matching split for $j_1$
- $s_2$ is not a matching split for $j_2$
- $j_1 (s_1)$
- $j_2 (s_2)$
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

fc

wfa
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)

\( s \)  

\( V \)  

\( j \) (s)  

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

- **EPC element**: `s` (AND split)
- **net fragment**: corresponding AND split

- **wfa**: `j (s)`
Step 1: OR join (wfa)

EPC element

Corresponding XOR split

Net fragment

\( wfa \)
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)

EPC element

s

XOR

\[ j(s) \]

net fragment

\[
\begin{array}{c}
EPC element \\
XOR \\
\end{array}
\]

\[
\begin{array}{c}
\text{corresponding XOR split} \\
\end{array}
\]

\[
\begin{array}{c}
\text{net fragment} \\
\end{array}
\]
Step 1: OR join (et)

EPC element

\[ EPC \text{ element} \]

\[ \text{s} \]

\[ \bigwedge \]

\[ j(s) \]

\[ \text{et} \]

net fragment

Corresponding AND split

\[ \text{corresponding AND split} \]

\[ \text{net fragment} \]

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)
Step 1: OR join (fc)

EPC element

\[ s \]

corresponding
AND split

\[ j(s) \]

fc

first come: at most one token gets through
(pending tokens may remain)

net fragment
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

straight conversion
Step 2: dummy style

needs a dummy transition

needs a dummy place

136
Example

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
Example

Steps

Sound?

Steps 1+2(+3)
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.