Methods for the specification and verification of business processes

MPB (6 cfu, 295AA)

Roberto Bruni

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

20 - Event-driven process chains
We overview EPC and the main challenges that arise when analysing them with Petri nets.

Ch.4.3, 6 of Business Process Management: Concepts, Languages, Architectures
An Event-driven Process Chain (EPC) is a particular type of flow-chart that can be used for configuring an Enterprise Resource Planning (ERP) implementation.

Supported by many tools (e.g. SAP R/3)

EPC Markup Language available (EPML) as interchange format
EPC overview

Important notation to model the domain aspects of business processes

Rather informal notation

EPC focus is on representing domain concepts and processes (not their formal aspects and technical realization)

It can be used to drive the modelling, analysis and redesign of business process
EPC origin

EPC method was originally developed by Wilhelm-August Scheer (early 1990’s)

Part of a holistic modelling approach called ARIS framework
(Architecture of Integrated Information Systems)
ARIS house (1999): three pillars and a roof...

Data  Control  Functions

Organization
...and three levels of abstraction each
...and three levels of abstraction each
EPC informally

An EPC is an “ordered” graph of events and functions

It provides various connectors that allow alternative and parallel execution of processes

The flow is specified by logical operators AND, XOR, OR

Simple, easy-to-understand notation
EPC ingredients:

**Event**

Any EPC diagram must start with **event(s)** and end with **event(s)**

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)

Graphical representation: hexagons
EPC ingredients:

**Function**

Any EPC diagram may involve several **functions**

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

Functions can be refined to other EPC

Graphical representation: rounded rectangles
EPC ingredients: Logical connectors

Any EPC diagram may involve several connectors

Elements used to describe the logical relationships between elements in the diagram

Branch, merge, fork, join

Graphical representation: circles (or also octagons)

\( \wedge \) AND, \( \lor \) OR, \( \oplus \) XOR
EPC ingredients: Control flow

Any EPC diagram may involve several control flow connections.

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

Graphical representation: dashed arrows.
EPC ingredients at a glance
EPC ingredients: Diagrams

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

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

Functions have exactly one incoming and one outgoing arc

The graph is weakly connected (no isolated nodes)

Connectors have either one incoming arc and multiple outgoing arcs or viceversa (multiple incoming arcs and one outgoing arc)
EPC ingredients: Diagrams

Other constraints are sometimes imposed

There is no arc between two events
There is no arc between two functions

Unique start / end event

No event is followed by a decision node (i.e. (X)OR-split)
EPC allowed connections
Three types of EPC objects can be used to model the control-flow aspect of a process: functions, events, and connectors. In a natural way, these types correspond to the BPMN activities, events, and gateways. However, EPCs do not allow for exceptions, and it supports only a limited set of connectors, which is shown by Fig. 4. Apart from the full set of connectors, this figure also shows an example process as an EPC, and it relates the object types to the workflow patterns explained in Section 2.2.

4.2 Transformation Challenges

A main challenge in EPCs is the semantics of the constructs that support the 'Simple Merge' and 'General Synchronizing Merge' patterns, viz. the XOR-join connector and the OR-join connector. Everybody agrees that the XOR-join connector should be enabled if one of its inputs is enabled, but this agreement is lacking in case more than one input is enabled. Some say that the XOR-join should be executed for every single enabled input, while others say that the connector should block if multiple inputs are enabled. An even bigger problem is the OR-join connector, for which a definitive semantics has lead to extensive discussions in literature and to different solutions, all of which fail for some EPCs [17,18,19]. As a result, not everybody will agree on a given mapping, as not everyone will agree with the semantics used by it.

Furthermore, an EPC allows for multiple start events and multiple final events, but not all combinations of these events are possible. Although the process designer might know the possible combinations, an EPC does not contain this information.
Other stuff

Other decorations / annotations for functions:

**Information, material, resource object:** represents objects in the real world that can be input data or output data for a function (rectangles linked to function boxes)

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

**Supporting system:** technical support (rectangles with vertical lines on its sides)
Other stuff
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”
Folder-passing semantics

Semantics

- State: Process folders
- Transition relation: Propagation of process folders
Folder-passing semantics: events
Folder-passing semantics: AND-split
Folder-passing semantics: XOR-split
Folder-passing semantics: XOR-join
Folder-semantics
in one slide

Figure 1: An EPC

Figure 2: The transition relation for the different nodes

27
A vicious circle

Figure 2 shows another EPC 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 $f_1$ and $f_2$. The two OR-joins will not propagate the process folders to the Inner events.

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 $\text{Inner}_1$ event. Since the $\text{Inner}_2$ 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 $f_1$ to the event $\text{Inner}_1$. Symmetrically, we can argue that the process folder from $f_2$ should be propagated to $\text{Inner}_2$. So, we have shown that the process folders should not be delayed at $f_1$ and $f_2$ according to the informal semantics of EPCs.

Is this the intended semantics of this EPC? Again, we will argue that it is not. We will argue that the OR-joins should not propagate the process folders from $f_1$ and $f_2$. To this end, we consider the OR-join before the $\text{Inner}_1$ event again. Since $\text{Inner}_2$ 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 $f_2$ should not be propagated. So, we know that the process folders should be delayed at $f_1$ and $f_2$ according to the informal semantics of EPCs.

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.
A vicious circle?

- c1
  - Inner1
    - Stop1
- c2
  - Inner2
    - Stop2
- c3
  - Inner3
    - Stop3
EPC semantics?

Little unanimity around the EPC semantics

Roughly described (verbal form) in the original publication by Scheer (1992)

Later, several attempts to define formal semantics (in many cases they end up attributing different meanings to the same EPC)

Discrepancies typically stem from the interpretation of (X)OR connectors (in particular, join case)
Other issues: unclear start, join/split balancing, alternation of events and functions
Problem with start events

A start event is an event with no incoming arc

A start event invokes a new execution of the process template

What if multiple start events occur?

Start events are mutually exclusive
(as if they were preceded by an implicit XOR split)
Problem with start events: solution

hypothetical / implicit split
Problem with alternation

Empirical studies have shown that middle and upper management people consider strict alternation between events and functions as too restrictive: they find it hard to identify the necessary events at the abstract level of process description they are working at.

It is safe to drop this requirement, as dummy events might always be added later, if needed.
Every join has a split

In theory, every join has at least one corresponding split (i.e. a split for which there is a path from either output to the input of the join)

**proof:**
we trace backward the paths leading to the join from start events; if the start events coincide there is a split node in the path; if start events differ, the candidate split is the implicit XOR
Problem with corresponding splits

The semantics of a join often depends on whether or not it has a corresponding split

But:
1) there can be more candidates to corresponding split
2) and they can have different type than the join

candidates of the same type of the join are called matching split

Some suggested to have a flag that denotes the corresponding split
Tagging corresponding splits

\[ s_1 \]

\[ s_2 \]

\[ j_1 \{s_1\} \]

\[ j_2 \{s_2\} \]
Problem with OR join

If an OR join has a matching split, the semantics is usually: “wait for the completion of all paths activated by the matching split”

If there is no matching split, some policy must be applied:

**wait-for-all**: wait for the completion of all *activated* paths (default semantics, because it coincides with that of a matched OR)

**first-come**: wait only for the path that is completed first and ignore the second

**every-time**: trigger the outgoing path on each completion (the outgoing path can be activated multiple times)

Some suggested to have different (trapezoid) symbols or suitable flags to distinguish the above cases and allow them all
Problem with XOR join

Similar considerations hold for the XOR join

If a XOR join has a matching split, the semantics is intuitive: “it blocks if both paths are activated and it is triggered by the completion of a single activated path”

If there is no matching split:
all feasible interpretations that do not involve blocking are already covered by the OR (wait-for-all, first-come, every-time) and \textbf{contradict the exclusivity} of the XOR
(a token from one path can be accepted only if we make sure that no second token will arrive via the other path)

Some suggest to just forbid the use of XOR in the unmatched case (the implicit start split is allowed as a valid match)
Translation of EPC to Petri nets
Idea

We transform EPC diagrams to Workflow nets

We exploit the formal semantics of nets to give unambiguous semantics to EPC diagrams

We apply the verification tools we have seen to check if the net is sound:
the EPC diagram is sound if its net is so
A note about the transformation

We first transform each event, function and connector separately in small net fragments.

When translating the control flow arcs we may then introduce other places / transitions to preserve the bipartite structure in the net (no arc allowed between two places, no arc allowed between two transitions).

We show two translations, depending on whether joins are decorated or not.
First attempt (decorated EPC)
EPC

Petri net

event

place
EPC

Petri net

AND split

net
EPC

OR split

Petri net

net
EPC

Petri net

XOR split

net
EPC

AND join

Petri net

net
EPC

Petri net

corresponding XOR/OR split

XOR join

ok

net
EPC

Petri net

corresponding
AND/OR split

deadlock!

dangling tokens!

XOR

XOR join

net
EPC

OR join with matched OR split

Petri net

matched part

ett
EPC

OR join
wait-for-all (unmatched)

Petri net

corresponding
split

wfa

net
EPC

OR join
wait-for-all
(unmatched)

Petri net

corresponding
AND split

net
EPC

OR join
wait-for-all
(unmatched)

Petri net

corresponding
XOR split

wfa

net
EPC

OR join
first-come
(unmatched)

Petri net

corresponding
split

net
EPC

OR join
first-come
(unmatched)

Petri net

corresponding
XOR split

ok

net
EPC

OR join
first-come
(unmatched)

Petri net

corresponding
AND split

dangling
token!
EPC

OR join every-time (unmatched)

Petri net

corresponding split

net
EPC

Petri net

corresponding
XOR split

OR join
every-time
(unmatched)

et

ok

net
EPC

OR join
every-time
(unmatched)

Petri net

corresponding
AND split

two
tokens!

net
Ill-formed net

Petri net

dummy transition
Ill-formed net

Petri net

dummy place
Exercise

Sound?
Fig. 13: Example of a modEPC
(first-come)
Exercise

Sound?
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`.

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.
Summary of problems

We need to decorate the EPC diagram
join decorated with matching/corresponding splits
OR-join decorated with policies

Split / join mismatch may induce unexpected behaviour

Possible introduction of dummy places and transitions
Second attempt
(no decoration available)
Simplified EPC

We rely on event / function alternation along paths in the diagram and also along paths between two connectors.

OR-connectors are not considered.
EPC 2 Petri nets: events and functions

- event
- function
- place
- transition

Definition 5: The formalization of the mapping is rather straightforward. Currently, we assume all the connectors to be of type join connectors of type split nodes. If the type of a join connector and the type of the output node do not agree, the connector is replaced by a small network. If the type of a transition (see Figure 4). If the type of a join connector is replaced by two or more arcs. For example, a join connector of type a place, i.e., a connector of type transitions. Figure 4 shows the rules that are used to map connectors onto Petri net correspond to the behavior of a connector in the event-driven process chain. Transitions net generated by correspond to functions or are the result of the translation of a connector.
EPC 2 Petri nets: 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
EPC 2 Petri nets: AND split

(event to functions) (function to events)
EPC 2 Petri nets: AND-join

(events to function) (functions to event)
EPC 2 Petri nets: XOR split

Figure 4: Mapping connectors onto places and transitions.

(event to functions)  (function to events)
EPC 2 Petri nets: XOR join

(events to function) (functions to event)
EPC 2 Petri nets: at a glance

- Event corresponds to functions or are the result of the translation of a connector. Each net generated by a transition (see Figure 4). If the type of a join connector and the type of the output node do not agree, the connector is replaced by a small network. If the type of a join connector agrees the type of the output node in the corresponding Petri net, the connector does not agree with the type of the output node in the Petri net, the connector is replaced by a small network. Otherwise, the connector is replaced by a split node.
EPC 2 nets: Example

In Table 1 it is assumed that connectors are only connected to functions and events, i.e., although it is possible to extend Table 1 with additional rules for connections between connectors, we use an alternative approach. Every arc connecting two connectors is replaced by an event and a function, i.e., fake events and functions are added to the event-driven process chain before the translation to a Petri net. Figure 5 illustrates the approach that is used to handle arcs in EPC.

The arc between the XOR-join (join connector of type XOR) and the AND-join (join connector of type AND) is replaced by function X and event X and three arcs. The arc between the AND-join and the XOR-split is also replaced by a function, an event and three arcs.

Figure 5: Arcs between connectors are replaced by events and functions before the event-driven process chain is mapped onto a Petri net.

Figure 6 shows the Petri net which corresponds to the event-driven process chain shown in Figure 1. Note that the arc between the two XOR connectors is replaced by an event and a function, and mapped onto an additional place and transition in the Petri net. In this case there was no real need to add these additional nodes. However, there are situations where adding events and functions is the only way to model the control flow properly.

It is easy to see that for any event-driven process chain satisfying the requirements in Definition 4, is a Petri net, i.e., and . Moreover, the Petri net is free-choice (see Definition 12).

(add dummy events and functions) (context-dependent translation)
From any EPC we derive a free-choice net

Moreover, if we add unique start / end events (and suitable transitions attached to them) the net is a workflow net
Exercise

Check it sound!
Exercise

Sound?

Figure 7: An erroneous event-driven process chain.

(remind to add dummy events and functions and to guarantee event/function alternation)
Relaxed soundness
(a third attempt)
Popularity vs superiority

EPC are a quite successful, semiformal notation

They lack a comprehensive and consistent syntax
They lack even more a corresponding semantics

You may **restrict the notation**, but people will prefer the more liberal (flexible) syntax and ignore the guidelines

You may **enrich the notation**, but people will dislike or misinterpret implementation policies
What are ultimately business process?

Graphical language to communicate concepts

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

Greatest common denominator of the people involved

Intuitive meaning
(verbatim description, no math involved)
Remember some good old friends

Chief Process Officer

Business engineer

Knowledge worker

Process designer

EPC

Process participants

Process responsible

System architect

WFnet

System developer
A secret not to tell

Ambiguity is useful in practice!

The more ways are to interpret a certain construct the more likely an agreement will be reached
A pragmatic consideration

Moreover

in the analysis phase
the participants may not be ready

to finalise the specification
and decide for the correct interpretation

Yet

it is important to find out flaws as soon as possible
Consequences

Ambiguous process description can constitute a major problem in the design phase

Therefore

we need to fix a formal representation that preserves all ambiguities
Problem

EPC is fine (widely adopted)

WF nets offer a useful tool

but

Soundness is too demanding at early stages
Relaxed soundness

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

Execution paths leading to unsound behaviour
can be used to infer potential mistakes in the EPC

If some unsound behaviour is possible
but enough sound paths exist
the process is called relaxed sound
A 3-steps approach
(keep it simple!)
Step 1: straightforward element map

<table>
<thead>
<tr>
<th>EPC</th>
<th>PN</th>
<th>EPC</th>
<th>PN</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="EPC Diagram" /></td>
<td><img src="image" alt="PN Diagram" /></td>
<td><img src="image" alt="EPC Diagram" /></td>
<td><img src="image" alt="PN Diagram" /></td>
</tr>
<tr>
<td><img src="image" alt="EPC Diagram" /></td>
<td><img src="image" alt="PN Diagram" /></td>
<td><img src="image" alt="EPC Diagram" /></td>
<td><img src="image" alt="PN Diagram" /></td>
</tr>
<tr>
<td><img src="image" alt="EPC Diagram" /></td>
<td><img src="image" alt="PN Diagram" /></td>
<td><img src="image" alt="EPC Diagram" /></td>
<td><img src="image" alt="PN Diagram" /></td>
</tr>
<tr>
<td><img src="image" alt="EPC Diagram" /></td>
<td><img src="image" alt="PN Diagram" /></td>
<td><img src="image" alt="EPC Diagram" /></td>
<td><img src="image" alt="PN Diagram" /></td>
</tr>
</tbody>
</table>

Note that the case that $E$ is reached twice if $F_1$ and $F_2$ occur sequentially has not been excluded.

Fig. 2. Transformation rules for an EPC into a place/transition net (rule 1)

Fig. 3. Transformation of the OR-Connector

To form a coherent Petri net the single modules are (automatically) connected as follows (rule 2): a) if input and output elements are different (place and transition) then the arcs are fused.
Step 2: element fusion

Step 1
Mapping EPC elements to PN-modules

Unification of elements (Case 1)

Step 2
Modul combination

Fusion of arcs (Case 2)

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.
Step 3: add unique start / end

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.

(sometimes XOR/AND can be preferred)
marks the termination of it. For example, the event not_ok triggers the function complaint whereas the event data revised marks the termination of complaint.

Furthermore, to describe more complex behaviour such as sequential, conditional, parallel, and iterative routing, connectors are introduced. These fall into two categories: splits and joins. In both we have AND, XOR and OR connectors.

Fig. 1. Handling of incoming goods
Relaxed Soundness of Business Processes

98

marks the termination of it. For example, the event not_ok triggers the function complaint whereas the event data revised marks the termination of complaint.

Furthermore, to describe more complex behaviour such as sequential, conditional, parallel, and iterative routing, connectors are introduced. These fall into two categories: splits and joins. In both we have AND, XOR and OR connectors.

Example
Example

Not sound!
Example

We can turn it to sound, but:
small changes in the net, turn big in EPC
Relaxed soundness: formally

**Definition:** A WF net is relaxed sound if every transition belongs to a firing sequence that starts in state $i$ and ends in state $o$.

$$\forall t \in T. \exists M, M'. i \rightarrow^* M \xrightarrow{t} M' \rightarrow^* o$$

(it is sound “enough”, in the sense that all transitions are covered by at least one sound execution)
Relaxed sound?
Relaxed soundness of business processes marks the termination of it. For example, the event not_ok triggers the function complaint whereas the event data revised marks the termination of complaint. Furthermore, to describe more complex behaviour such as sequential, conditional, parallel, and iterative routing, connectors are introduced. These fall into two categories: splits and joins. In both we have AND, XOR and OR connectors.

Fig. 1. Handling of incoming goods

Applying the proposed rules 1 to 3, an EPC is transformed into a WF net. This transformation is unique, in the sense that to each EPC belongs exactly one WF net. An example for such a transformation is shown in Fig. 5. Here the EPC from Fig. 1 has been transformed into a WF net. For convenience we surrounded the Petri net-modules which correspond to the routing constructs of the EPC with dotted rectangles.

Transition t10_AND-Join and the sink place o have been added due to rule 3. Transition t10_AND-Join corresponds to an AND connector which complements the last connector on the paths from the end events E12 and E20, namely connector C12. Transition t10_AND-Join bundles the different path and leads to the sink place o.
Relaxed soundness of business processes

Relaxed soundness of business processes

... marks the termination of it. For example, the event `not_ok` triggers the function `complaint` whereas the event `data revised` marks the termination of `complaint`.

Furthermore, to describe more complex behaviour such as sequential, conditional, parallel, and iterative routing, connectors are introduced. These fall into two categories: splits and joins. In both we have AND, XOR and OR connectors.

Example

Relaxed sound?
Example

Relaxed sound?
Example

Not relaxed sound (as WF net)!
But relaxed sound as EPC
(all nodes covered by some sound execution)
Pros and Cons

If the WF net is not sound:
there are transitions that are not contained in any sound firing sequence

Hence their EPC counterparts need improvements

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

No equivalent characterization is known
that is more convenient to check

Open research problem…