Methods for the specification and verification of business processes

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.
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

Rather informal notation
simple and easy-to-understand

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...
...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.
Events

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.
Functions

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
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)
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

Event

Function

Connectors

Combined Connectors (sample)

Control Flow
EPC 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 vice versa (multiple incoming arcs and one outgoing arc)
EPC ingredients: Diagrams

Other constraints are sometimes imposed

Unique start / end event

No arc between two events
No arc between two functions

No event is followed by a decision node (i.e. (X)OR-split)
EPC allowed connections

<table>
<thead>
<tr>
<th>Event Connection</th>
<th>Activity Connection</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AND</strong></td>
<td></td>
</tr>
<tr>
<td>triggering Events</td>
<td>triggered Events</td>
</tr>
<tr>
<td>Event 1</td>
<td>Activity</td>
</tr>
<tr>
<td>Event 2</td>
<td>Event 1</td>
</tr>
<tr>
<td>Activity</td>
<td>Event 2</td>
</tr>
<tr>
<td>Event 1</td>
<td>Activity</td>
</tr>
<tr>
<td>Event 2</td>
<td>Event 1</td>
</tr>
<tr>
<td>Activity</td>
<td>Event 2</td>
</tr>
<tr>
<td><strong>OR</strong></td>
<td></td>
</tr>
<tr>
<td>triggering Events</td>
<td>triggered Events</td>
</tr>
<tr>
<td>Event 1</td>
<td>Activity</td>
</tr>
<tr>
<td>Event 2</td>
<td>Event 1</td>
</tr>
<tr>
<td>Activity</td>
<td>Event 2</td>
</tr>
<tr>
<td>Event 1</td>
<td>Activity</td>
</tr>
<tr>
<td>Event 2</td>
<td>Event 1</td>
</tr>
<tr>
<td>Activity</td>
<td>Event 2</td>
</tr>
<tr>
<td><strong>XOR</strong></td>
<td></td>
</tr>
<tr>
<td>triggering Events</td>
<td>triggered Events</td>
</tr>
<tr>
<td>Event 1</td>
<td>Activity</td>
</tr>
<tr>
<td>Event 2</td>
<td>Event 1</td>
</tr>
<tr>
<td>Activity</td>
<td>Event 2</td>
</tr>
<tr>
<td>Event 1</td>
<td>Activity</td>
</tr>
<tr>
<td>Event 2</td>
<td>Event 1</td>
</tr>
<tr>
<td>Activity</td>
<td>Event 2</td>
</tr>
</tbody>
</table>

*not allowed*
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 inputs is enabled. Some say that the XOR-join connector 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 EPCs.
Other annotations for functions

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)

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)
EPC Diagram always starts with Event

Event describes what circumstances a function or a process works or which state a function or process results in.
Organization unit assignments show the connection between an organization unit and the function it is responsible for.

Organization units determine which person or organization within the structure of an enterprise is responsible for a specific function.

Function describes the transformations from an initial state to a resulting state.
Information, material, or resource objects portray objects in the real world, for example business objects, entities, etc., which can be input data serving as the basis for a function, or output data produced by a function.
XOR operator corresponds to making decision of which path to choose among several control flows.

A control flow connects events with functions, process paths, or logical connectors.
This object represents information system.

Information flows show the connection between functions and input or output data, upon which the function reads changes or writes.
EPC Diagram always ends with Event
EPC Diagram always starts with Event

Event describes what circumstances a function or a process works or which state a function or process results in.

Information, material, or resource objects portray objects in the real world, for example business objects, entities, etc., which can be input data serving as the basis for a function, or output data produced by a function.

Organization unit assignments show the connection between an organization unit and the function it is responsible for.

Organization units determine which person or organization within the structure of an enterprise is responsible for a specific function.

Function describes the transformers from an initial state to a resulting state.

XOR operator corresponds to making decision of which path to choose among several control flows.

A control flow connects events with functions, process paths, or logical connectors.

Information flows show the connection between functions and input or output data, upon which the function reads changes or writes.
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”
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

dead path elimination

dead path elimination
Folder-passing semantics: XOR-join
Folder-semantics in one slide

Figure 1: An EPC

Figure 2: The transition relation for the different nodes
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.

Start1

Stop1

Inner1

f'1

f1

∧

Inner2

f2

f'2

∧

Stop2

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 \( Inner_1 \) event. Since the \( Inner_2 \) 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 \( Inner_1 \). Symmetrically, we can argue that the process folder from \( f_2 \) should be propagated to \( Inner_2 \). 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 EPCs? 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 \( Inner_1 \) event again. Since \( Inner_2 \) 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.
EPC 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)

Discrepancies typically stem from the interpretation
of (X)OR connectors (in particular, join case)
Other issues: unclear start,
join/split balancing,
alternation between 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?

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

hypothetical / implicit split

A

XOR

B
Problem with alternation

From empirical studies:
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

Solution:
It is safe to drop the requirement about alternation
(dummy events might always be added later)
Every join has a split

observation:
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 sketch:
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 the nature of the 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 matching split)

**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
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 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)
Sound EPC diagrams

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

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.
Translation of EPC to Petri nets
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 different translations, depending on whether joins are decorated or not.
First attempt
(decorated EPC)
EPC  Petri net

event  place
EPC

Petri net

function

transition
EPC

AND split

Petri net

net
EPC  

Petri net  

XOR split  

net
EPC

OR split

Petri net

net
EPC

OR split

Petri net

xor
and
net
EPC

OR split

Petri net

xor

and

net
EPC

AND join

Petri net

net
EPC

Petri net

corresponding
split

XOR join

net
EPC

Petri net

XOR join

most general case

net
EPC

Petri net

corresponding XOR/OR split

XOR join

ok net
EPC

Petri net

corresponding AND/OR split

deadlock!

XOR join

net
better to have a corresponding XOR split!
EPC

OR join

Petri net

split

corresponding

net
EPC

Petri net

OR join
with
matched OR split

matching OR split

net
EPC

mismatched corresponding split:
most general case

Petri net

OR join
wait-for-all
(mismatched)
EPC

Petri net

corresponding
AND split

OR join
wait-for-all
(mismatched)
EPC

OR join
wait-for-all
(mismatched)

Petri net

corresponding
XOR split

wfa

net
EPC

Petri net

wfa works well with any corresponding split

OR join

wait-for-all (mismatched)
EPC

mismatched corresponding split: most general case

Petri net

OR join
first-come (mismatched)
EPC

OR join
first-come
(unmatched)

Petri net

corresponding
XOR split

ok

net
EPC

OR join
first-come (unmatched)

Petri net

corresponding
AND split

pending token!
EPC  Petri net

fc:
better to have
a corresponding
XOR split!

OR join
first-come
(mismatched)
EPC

mismatched corresponding split: most general case

Petri net

OR join every-time (mismatched)

net
EPC

OR join every-time (mismatched)

Petri net

corresponding XOR split

ok
EPC

Petri net

corresponding
AND split

OR join every-time (unmatched)
OR join every-time (mismatched)

better to have a corresponding XOR split!
split

join

better to have a corresp. XOR split

corresp. split

matched OR split
corresp. split: wfa
corresp. split: fc

corresp. split: et

better to have a corresp. XOR split
Ill-formed net

Petri net

dummy transition
Ill-formed net

Petri net

dummy place
Example
Fig. 13: Example of a modEPC implicit XOR

Example
Example

Fig. 13: Example of a modEPC

implicit XOR
Example

Fig. 13: Example of a modEPC
Example

Fig. 13: Example of a modEPC
Example

Fig. 13: Example of a modEPC
Exercise

Sound?

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

Sound?
Exercise

Sound?
Exercise

Sound?
Exercise

Sound?
Exercise

Sound?
Exercise

Sound?
Exercise

Sound?
Exercise

Sound?
Exercise

Fig. 13: Example of a modEPC

(forest-come)

Sound?
Exercise

Sound?
Exercise

Sound?

Fig. 13: Example of a modEPC

(first-come)
Exercise

Sound?
Exercise

Sound?
Fig. 13: Example of a modEPC (first-come)
ZOOM IN
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
joins must be decorated with matching/corresponding splits
mismatched OR-joins must be decorated with policies

Split / join mismatch may induce unexpected behaviour

Possible introduction of dummy places and transitions
Second attempt
(no decoration required)
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

Let EPC 2 Petri nets:

Assume the formalization of the mapping is rather straightforward. Based on this assumption the moment, we assume all the connectors to be of type join connectors of type split connectors. The behavior of a connector of type transitions. Figure 4 shows the rules that are used to map connectors onto Petri net. A connector of type a place, i.e., a connector of type transitions. If the type of a connector is replaced by two or more arcs. For example, a join connector of type a number or arcs.

If the type of a connector does not agree with the type of the output node in the Petri net, the connector is replaced by a small network. If the type of a connector agrees the type of the output node do not agree, the connector is replaced by a small network. If the type of a connector is replaced by a small network of places and transitions. The translation of connectors is much more complex. A connector may correspond to a number of arcs in the Petri net or to a small network of places and transitions. Each model the behavior of a connector in the event-driven process chain. Transitions
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

Figure 4: Mapping connectors onto places and transitions.

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

The arc between the XOR-join (join connector of type \(V\)) and the AND-join (join connector of type \(V\)) 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., \( \). Moreover, the Petri net is free-choice (see Definition 12).
Outcome

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
Figure 1: Modeling of a business process, using event-driven process chains.

Figure 6: The event-driven process chain of Figure 1, mapped onto a Petri net.

Exercise

Check it sound!
Exercise

Sound?

(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 (verbal description, no math involved)
Remember some good old friends

- Chief Process Officer
- Business engineer
- Knowledge worker
- Process designer
- Process participants
- Process responsible
- System architect
- System developer
- EPC
- WFnet
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 descriptions arise 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

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

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

Fig. 3. Transformation of the OR-Connector
Step 2: element fusion

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

The paths finally join. The EPC syntax rules state that: For every two elements there is a path
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.

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 depends on the relation of the corresponding events in the EPC.
Example

Sound?
Example
Example

Not sound!
We can turn it to sound, but: small changes in the net, can be problematic 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 behavior 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

Relaxed sound?
Example

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

Relaxed Soundness of Business Processes
Pros and Cons

If the WF net is not relaxed sound:
there are transitions that are not part of a
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…