Business Processes Modelling

**MPB (6 cfu, 295AA)**

Roberto Bruni

[http://www.di.unipi.it/~bruni](http://www.di.unipi.it/~bruni)

06 - Event-driven Process Chains
We overview the EPC notation

Ch.4.3 of Business Process Management: Concepts, Languages, Architectures
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 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
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
(verbial description, no math involved)
EPC informally

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

Event

Function

Connectors: $\land$, $\lor$, XOR

Control Flow
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)

AND \( \wedge \)  \hspace{1cm} XOR \( \times \)  \hspace{1cm} OR \( \lor \)

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: requirements

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

AND
XOR
OR

AND
XOR
OR
EPC: Example ($yEd$)

- $\wedge$ = AND
- $X$ = XOR
- $\lor$ = OR

Flowchart:
- Start
  - Travel request
    - XOR
      - AND
        - Book flight
        - Book hotel
      - AND
        - Travel planned
          - Ask confirmation
            - XOR
              - Confirm
                - End ok
              - Cancel
                - End null
            - Change request
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
add dummy nodes to guarantee no event be followed by a decision node (\((X)\text{OR}\)-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.

End1 End2

End1 End2

AND? XOR? OR?
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)
A taste of EPML

EPC markup language (EPML): an XML-based interchange format for event-driven process chains (EPC)

Jan Mendling · Markus Nüttgens

Published online: 22 October 2005
© Springer-Verlag 2005
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”
Folder-passing semantics

The current state of the process is determined by placing folders over the diagram.

A transition relation explains how to move from one state to the next state.

The transition relation is possibly nondeterministic.
Folder-passing semantics: events
Folder-passing semantics: functions

a function → a function
Folder-passing semantics: AND-split
Folder-passing semantics: AND-join
Folder-passing semantics: XOR-split
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
Folder-passing semantics: XOR-join
Folder-passing semantics?

How can we infer the absence of folders?

When and how should such information be propagated?
Folder-passing semantics: OR-split
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
Folder-passing semantics: OR-join?
A vicious circle?

A conceptual problem

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 Inner1 event. Since the Inner2 event will never occur, we know that no process folder will ever arrive at the other incoming arc of the OR-join. So, according to the informal semantics, the OR-join should propagate the process folder from $f_1$ to the event Inner1. Symmetrically, we can argue that the process folder from $f_2$ should be propagated to Inner2. 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 Inner1 event again. Since Inner2 will eventually occur, we know that eventually there will be a process folder arriving at the second incoming arc. According to the informal semantics, this implies that the OR-join should wait with the propagation of the process folder until the second folder arrives. Symmetrically, we can argue that the process folder from $f_2$ 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.
Decorated EPC

To remove ambiguous behaviour for join connectors, designers can further annotate EPC diagrams

In particular we require to know:

- **corresponding split**
  which node separated the flows we are joining
  (in the case of XOR/OR join)

- **applicable policy**
  how to trigger outgoing flow
  (avoid OR join ambiguous behaviour)
A candidate split for a join node is any split node whose outputs are connected to the inputs of the join.

\[ s_1 \text{ is a candidate split for } j_1 \]

\[ s_1 \text{ and } s_2 \text{ are candidate splits for } j_2 \]
A corresponding split for a join node is a chosen candidate split

we choose s1 as a corresponding split for j1
we choose s2 as a corresponding split for j2

(we tag each join with its corresponding 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: policies

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

- Start
- Visit farmhouse
- XOR
  - Guide available
  - Visit winery
    - AND
      - Buy products
      - OR
        - OR
          - Pay
          - End
  - Visit animals
    - OR
      - Dinner
      - 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

et: every-time
fc: first-come
work well with corresponding XOR split
**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)
EPC Sample Diagrams
Example

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 led 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.
Example
Example
Example
Example
Example

Fig. 4.38. Sample function flow

Fig. 4.39. Example of extended event-driven process chain
Example

Fig. 1 Event-driven process chains representing the waterfall model for software engineering
Exercises

Transfer the following verbal description into an EPC

a) When creating a cost centre (CC), the controlling department specifies the validity period of the cost centre, and then assigns it to a cost centre group.

b) Once the cost centres are assigned to the standard hierarchy, the basic data of the cost centre (CC name, person in charge, currency and category) is simultaneously determined.

c) Thereafter the controlling department assigns the organizational units (business area, profit centre or both) to the cost centres.

d) Once the organizational units are assigned, the controlling department determines the control indicators. Finally, the controlling department saves the captured data and, as a result, the cost centre is created.