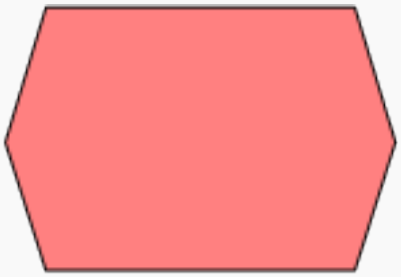
Event

From EPC Standard

The printable version is no longer supported and may have rendering errors. Please update your browser bookmarks and please use the default browser print function instead.

Event	
Graphical Notation	
	
IsSubClassOf	IsSubClassOf::Process element
Successors	hasSuccessor::Function, hasSuccessor::AND Operator
Predecessors	hasPredecessor::Function, hasPredecessor::Operator
HasIncomingControlFlow	hasIncomingControlFlow::0, hasIncomingControlFlow::1
HasOutgoingControlFlow	hasOutgoingControlFlow::0, hasOutgoingControlFlow::1
HasResource	hasResource::0
HasAttribute	hasAttribute::0, hasAttribute::n
Edit the Properties	

□

Contents

Brief Information

This is an autogenerated section!

You are not able to edit this information by hand, but by edit the Form (and therefore the properties) of

this page. Please refer to the Edit the properties link at the bottom of the info box. {{#show: Event | ?Is a | Intro=The Event is a }}. {{#show: Event | ?contains | Intro=It contains }}. {{#show: Event | ?hasSuccessor | Intro=Possible succeeding element(s) is/are }}. {{#show: Event | ?hasPredecessor | Intro=Previous element(s) can be }}. {{#show: Event | ?hasIncomingControlFlow | Intro=The cardinalities are | Outro= (incoming)}} {{#show: Event | ?hasOutgoingControlFlow | Intro=and | Outro= (outgoing) respectively }}. {{#show: Event | ?refersTo | Intro=The Event refers to }}. {{#show: Event | ?attachedTo | Intro=The Event is attached to a }}.

Syntax

The EPC Syntax requires an event either to be preceded by an operator or a function. In contrast, an event may be followed by a function or an AND Operator. An event is linked to its predecessor and successor via a control flow arc. The control flow arc cannot connect two events directly.[1], [2], [3] An event with no incoming arc is called “start event”, while an event with no outgoing arc is called “end event”.[4] The requirements of a well-formed EPC-model demand it to contain at least one start-event and one end-event.[3]

Semantic

In contrast to the function element, the event is a passive element since it represents a change of state but does not cause it. Furthermore it can characterize the conditions and the results of an activity, which in turn triggers the next function.[5], [6], [7], [8], [9], [10] Due to the semantics of events as passive elements, they lack the ability to determine the function(s) that should follow.[11] Therefore an event may be followed by a function or an AND Operator, but not by an OR or XOR Operator.[3] In the literature there is an ongoing debate on state representation in EPCs. [11] Some authors say that the current process state is represented by events[12], while others identify states with control flow arcs[13].

Semantic Representation

An event E is a part of an EPC = (E, F, P,C, l, A), for which E is defined: An element of E is called event. E ≠ ∅ E is a pairwise disjoint and finite set: E ∩ F= ∅, E ∩ C =∅ 1

An event also is a node N, being part of N = E ∪ F ∪ P ∪ C. [14]

An EPC begins with an amount of start events and ends with an amount of end events. Every other event is called intermediate event. An event is connected to other nodes (•e and e•) by incoming and outgoing arcs. The following subsets are defined:

- Es= {e ∈ E | |•e| = 0 ∧ |e•| = 1} being the set of start events
- Eint = {e ∈ E | |•e| = 1 ∧ |e•| = 1} being the set of intermediate events
- Ee = {e ∈ E | |•e| = 1| ∧ e•| = 0} being the set of end events
- CEF = {c ∈ C | ∗c c ⊆ E ∧ c c* ⊆ (F ∪ P)} as the set of event-function connectors. [15], [16]
- CFE = {c ∈ C | *c ⊆ (F ∪ P) ∧ c* ⊆ E} as a set of function-event (fe)-operators.
- nin = {(x,n) | x ∈ N ∧ (x,n) ∈ A) as a set of incoming control flow edges.
- nout = {(n,y) | y ∈ N ∧ (n,y) ∈ A) as a set of outcoming control flow edges.

Following requirements are made on events so an EPC can be called relaxed syntactically correct:

- ∀ n ∈ N : ∃e1 ∈ Es, e2 ∈ Ee such that e1 → n → ee the EPC is a direct and coherent graph →

Functions and events (maybe linked by connectors) should alternate along the control flow

- |Es ∪ Ps| ≥ 1 ∧ |Ee ∪ Pe| ≥ 1. There is at least one start node and one end node in an EPC
- ∀ e ∈ E : |•e| ≤ 1 ∧ |e•| ≤ 1 Events have at most one incoming and one outgoing arc

This implies that Es, Eint, and Ee partition E

- ∀ e ∈ E : •e ⊆ (F ∪ P ∪ CFE) (F ∪ P ∪ CEF)

Events must have function, process interface, or fe-connector nodes in the preset and function, process interface, or ef-connector nodes in the postset . ALSO Events must neither have an XOR, nor an OR connector in the postset . If •es ≠ ∅ (source node) and •ee ≠ ∅ (sink node) and every node n ∈ N is on a path from estart to efinal, a EPC is called regular. [17]

Linguistic Correctness

To satisfy the demands of pragmatic correctness every label of the model elements should follow a specified naming convention. In an EPC events are representing the change of a process state, so the linguistic correctness requires the label to be created from a substantive and a verb in the past participle form. The following table shows examples for one EPC modelled in German and one modelled in English.[3]

Language	Rule	Example
English	Substantive(s) + Verb in Past Participle	"Order processed"
German	Substantive(s) + Verb in Past Participle	"Bestellung bearbeitet"

To improve the ease of understanding the name of an event should reflect its characteristic as a point in time. Events should be named in a way that each event is produced by the preceding activity and that the same event triggers the execution of the next activity.[5]

XML Representation

Edit the XML Code

<source lang="xml">

<xs:complexType name="typeEvent"> <xs:sequence> <xs:element name="documentation" type="xs:anyType" minOccurs="0"/> <xs:element name="toolInfo" type="xs:anyType" minOccurs="0"/> <xs:element name="name" type="xs:string"/> <xs:element name="description" type="xs:string" minOccurs="0"/> <xs:choice minOccurs="0"> <xs:element name="graphics" type="epml:typeGraphics"/> </xs:choice> <xs:choice minOccurs="0"> <xs:element name="syntaxInfo"> <xs:complexType> <xs:attribute name="implicitType"> <xs:simpleType> <xs:restriction base="xs:string"> <xs:enumeration value="innerEvent"/> <xs:enumeration value="startEvent"/> <xs:enumeration value="endEvent"/> </xs:restriction> </xs:simpleType> </xs:attribute> </xs:complexType> </xs:element> </xs:choice> <xs:choice minOccurs="0" maxOccurs="unbounded"> <xs:element name="attribute" type="epml:typeAttribute"/> </xs:choice> </xs:sequence> <xs:attribute name="id" type="xs:positiveInteger" use="required"/> <xs:attribute name="defRef" type="xs:positiveInteger" use="optional"/> </xs:complexType> </source>

References

2. S. Appel, P. Kleber, S. Frischbier, T. Freudenreich, and A. Buchmann, “Modeling and execution of event stream processing in business processes,” Inf. Syst., vol. 46, pp. 140–156, 2014.
4. J. Dehnert, J. Dehnert, P. Rittgen, and P. Rittgen, “Relaxed soundness of business processes,” Lect. notes Comput. Sci., pp. 157–170, 2001.

6. M. Fellmann, S. Bittmann, A. Karhof, C. Stolze, and O. Thomas, “Do We Need a Standard for EPC Modelling? The State of Syntactic, Semantic and Pragmatic Quality,” in 5th International Workshop on Enterprise Modelling and Information Systems Architectures (EMISA), 2013, pp. 103–116.
8. R. Dijkman, “Diagnosing differences between business process models,” Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 5240 LNCS, pp. 261–277, 2008.
10. A. W. Scheer, O. Thomas, and O. Adam, Process Modeling using Event-Driven Process Chains. 2005.
12. J. Mendling, G. Neumann, and M. Nüttgens, “Yet another event-driven process chain,” in Lecture Notes in Computer Science, 2005, vol. 3649, p. 428.
14. F. Gottschalk, W. M. P. van der Aalst, and M. H. Jansen-Vullers, “Merging Event-Driven Process Chains,” pp. 418–426, 2008.
16. O. Kopp, M. Wieland, and F. Leymann, “External and Internal Events in EPCs : e2EPCs,” 2nd Int. Work. event-driven Bus. Process Manag., pp. 381–392, 2010.
18. A.-W. Scheer, M. Nüttgens, and V. Zimmermann, “Objektorientierte Ereignisgesteuerte Prozeßkette (oEPK): Methode und Anwendung,” Veröffentlichungen des Instituts für Wirtschaftsinformatik (IWi), Univ. des Saarlande, no. 141, 1997.
20. B. List and B. Korherr, “A UML 2 Profile for Business Process Modelling *,” vol. 205, pp. 85–96, 2005.
22. J. Mendling, M. Moser, and G. Neumann, “Transformation of yEPC Business Process Models to YAWL,” 2006 ACM Symp. Appl. Comput., pp. 1262–1266, 2006.
24. G. Keller, M. Nüttgens, and A.-W. Scheer, “Semantische Prozeßmodellierung auf der Grundlage „Ereignisgesteuerter Prozeßketten (EPK)“,” Veröffentlichungen des Instituts für Wirtschaftsinformatik (IWi), Univ. des Saarlandes, no. 89, 1992.
26. E. Kindler, “On the semantics of EPCs: Resolving the vicious circle,” Data Knowl. Eng., vol. 56, no. 1, pp. 23–40, 2006.
28. Mendling: Event Driven Process Chains - Metrics for Process Models, Volume 6 of the series Lecture Notes in Business Information Processing, pp. 17-57, 2009.
30. M. Nüttgens, F. J. Rump "Syntax und Semantik Ereignisgesteuerter Prozessketten", Prozessorientierte Methoden und Werkzeuge für die Entwicklung von Informationssystemen - Promise 2002, P.69
32. K. van Hee, O. Oanea, and N. Sidorova, "Colored Petri Nets to Verify Extended Event-Driven Process Chains", OTM Confederated International Conferences "On the Move to Meaningful Internet Systems", 2005, pp. 183-201.
34. Van der Aalst, "Formalization and verification of event-driven process chains" Information and Software Technology 41, 1999, pp. 639-650.

Category:
Meta Model