EPC Modelling based on Implicit Arc Types

Jan Mendling, Markus Nüttgens

Universität Trier
Wirtschaftsinformatik II,
Postfach 3825, D-54286 Trier
mendling@web.de, markus@nuettgens.de

Abstract: Event Driven Process Chains (EPC) are commonly used for the modelling of business processes. As modelling is decentralised to personnel not familiar with the formal aspects of this method, syntax checks are needed to avoid invalid models. This paper presents the concept of implicit element and arc types. It can be used both to support modellers in the process of building models and to check entire models. It is especially helpful to avoid closure calculation for connector type consistency constraints.

1. EPC and Business Process Modelling

1.1. Modelling with EPC between Intuition and Validity

Event Driven Process Chains (EPC) have been developed to model business processes on a conceptual level [KNS92]. Both in academics as integral part of the ARIS concept [Sc00] and in practice with SAP AG using them for their SAP reference model [Ke99], they have reached a wide-spread use. A major advantage of EPCs is their ability to express processes in an intuitive way. Thus, they are very often used for the documentation and management of business processes.

Despite their popularity and intuition, the syntax of EPCs has to meet a variety of validity rules concerning the sequence of different elements. This turns out to be a problem when organizations decentralize and delegate their business process modelling activities to the departments concerned. On the one hand, the personnel involved might be little experienced in assessing the formal validity of the models they produce. One the other hand, models have to be valid in order to be reused in workflow management systems. As a consequence two questions arise: Firstly, which concepts can be used to guide the modeller during the process of modelling to avoid syntactically invalid modelling. Secondly, how can the validity of a given EPC model checked in an efficient way. This paper addresses these two problems and presents a concept called “implicit arc types” as a solution.

1.2. EPC Syntax Related Work

Most of the formal contributions on EPCs have been focused on semantics, especially on the semantics of OR connectors. The translation of EPC process models to Petri Nets plays an important role in this context. Examples of this research can be found in Chen/Scheer [CS94], Rodenhagen [Ro97], Langner/Schneider/Wehler [LSW98], van der Aalst [Aa99], Rittgen [Ri00], and Dehnert [De02]. A major point of discussion is the "non-locality" of join-connectors [ADK02]. This paper will present a syntax related work based on the formal syntax definition of EPCs in [NR02]. Therefore we give a brief survey of syntax related work before presenting definitions in the second section.

In Keller/Nüttgens/Scheer the EPC is introduced [KNS92] to represent temporal and logical dependencies in business processes. Elements of EPCs may be of function type (active elements), event type (passive elements), or of one of the three connector types AND, OR, or XOR. These objects are linked via control flow arcs. Connectors may be split or join operators, starting either with function(s) or event(s). These four combinations are discussed for the three connectors resulting in twelve possibilities. Split OR and Split XOR are prohibited after events, due to the latter being unable to decide which following functions to choose. Based on practical experience with the SAP Reference model, process interfaces and hierarchical functions are introduced as additional element types of EPCs [KM94]. These two elements permit to link different EPC models: process interfaces can be used to refer from the end of a process to a following process, hierarchical functions allow to define macro-processes with the help of sub-processes. Keller [Ke99] and Rump [99] provide a formal approach defining the EPC syntax. Based on this, Nüttgens/Rump [NR02] distinguish the concepts of a flat EPC Schema and a hierarchical EPC Schema. A flat EPC Schema is defined as a directed and coherent graph with cardinality and type constraints. A hierarchical EPC Schema is a set of flat or hierarchical EPC Schemata. Hierarchical EPC Schemata consist of flat EPC Schemata and a hierarchy relation linking either a function or a process interface to another EPC Schema. Fig. 1 shows a hierarchical EPC Schema consisting of two processes, which are linked via a hierarchical relation attached to the process interface "To Design Process".

Type consistency of consecutive connectors poses a major problem for these definitions, because the EPC graph has to be traversed to find and check all non-connector ancestors and descendant elements. In section two we will introduce explicit and implicit element and arc types, and define EPCs with the help of them. The advantages of such a definition are presented in section three, where it is described how formally valid business process modelling with EPCs can be granted.

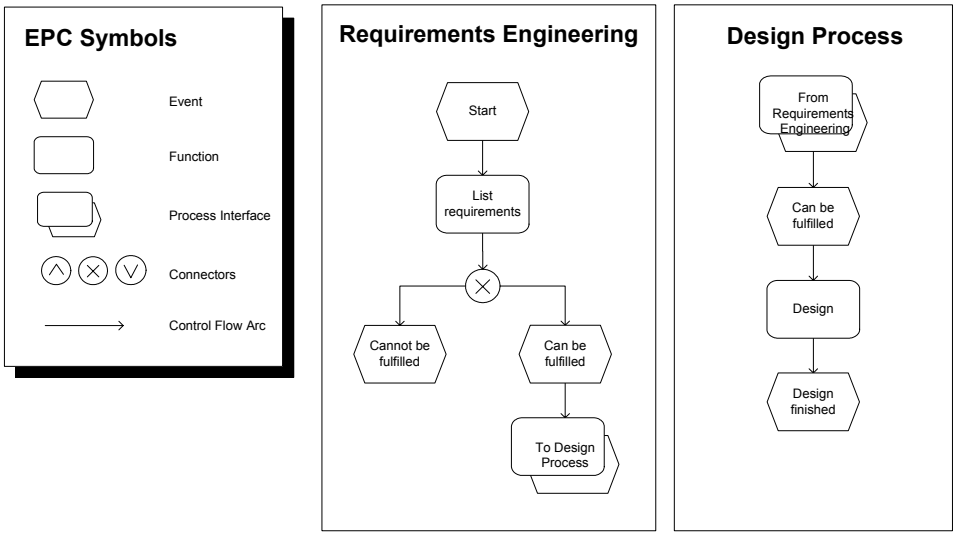

Fig. 1. EPC example of a simple requirements engineering process. The connector represents an "exclusive or". After "Can be fulfilled" a process interface links to the design process.

2. EPC Syntax Objects

2.1. Explicit and Implicit Element Types

The distinction of explicit and implicit element types is rooted in two different perspectives on business process modelling, the perspective of the modeller and the perspective of verification. The task of the modeller is to compose a structure from a given set of symbols that is able to represent concepts of the domain in a pragmatic and abbreviated way. This set of symbols modelling is provided by a business process modelling tool. Such EPC symbols usually refer to the original definition of Keller/Nüttgens/Scheer [KNS92] distinguishing event type E, function type F, process interface P, connector AND, connector OR, and connector XOR. We refer to them as explicit element types EXPL being mutually disjoint:

$$\text{EXPL} = \text{E} \cup \text{F} \cup \text{P} \cup \text{AND} \cup \text{OR} \cup \text{XOR}. \quad (1)$$

Implicit element types relate to the perspective of syntactical verification. They represent disjoint specialisations of corresponding explicit element types. Each implicit element type captures a specific constellation in which an explicit element type may occur. Each of these implicit roles implies different restrictions on the set of allowed ancestors and descendants, and their cardinality. Fig. 2 presents explicit and corresponding implicit element types: Events E may be Start Events E_S, Inner Event E_{Int} or End Event E_E. For process interfaces P can be used as start and end symbols. Connectors can be either joins

or splits. When they have (transitive) event ancestors, their (transitive) descendants have to be functions. When they have (transitive) function ancestors, their (transitive) descendants have to be events. For OR- and XOR-connectors Event-Function-Splits are forbidden [KNS92].

Explicit Element Type	**Implicit Element Types**
Event E	Start Event E_S Inner Event E_{Int} End Event E_E
Function F	Function F
Process Interface P	Start ProcessInterface P_S End ProcessInterface P_E
Connector AND	Event-Function-Split AND_{EFS} Event-Function-Join AND_{EFJ} Function-Event-Split AND_{FES} Function-Event-Join AND_{FEJ}
Connector OR	Event-Function-Join OR_{EFJ} Function-Event-Split OR_{FES} Function-Event-Join OR_{FEJ}
Connector XOR	Event-Function-Join XOR_{EFJ} Function-Event-Split XOR_{FES} Function-Event-Join XOR_{FEJ}

Fig. 2. Explicit and corresponding implicit element types.

2.2. Explicit and Implicit Arc Types

Analogously to the distinction between explicit and implicit element types, we define implicit arc types as a partition of the control flow arc (explicit arc type) [Me03]. Implicit arc types are subsets of the product of implicit element types. Fig. 3 presents which arcs are allowed from and to implicit element types. The 16x16 matrix shows 100 implicit arc types which are permitted. The distribution of them in this matrix suggests the distinction between two different groups of implicit arcs. We will refer to them as Function-Event-Arcs FEA and Event-Function-Arcs EFA. In order to define these two kinds of arcs, a grouping of implicit element types into Event-Types (from), Function-Types (from), Event-Types (to), and Function-Types (to) is needed:

$$\text{Event Types (from) } ET_{from} = E_S \cup E_{Int} \cup AND_{EFS} \cup AND_{EFJ} \cup OR_{EFJ} \cup XOR_{EFJ}\ , \quad (2)$$

$$\text{Function Types (from) } FT_{from} = F \cup P_S \cup AND_{FES} \cup AND_{FEJ} \cup OR_{FES} \cup OR_{FEJ} \cup XOR_{FES} \cup XOR_{FEJ}\ , \quad (3)$$

$$\text{Event Types (to) } ET_{to} = E_{Int} \cup E_E \cup AND_{EFS} \cup AND_{EFJ} \cup OR_{EFJ} \cup XOR_{EFJ}\ , \quad (4)$$

$$\text{Function Types (to) } FT_{to} = F \cup P_E \cup AND_{EFS} \cup AND_{EFJ} \cup OR_{EFJ} \cup XOR_{EFJ}\ . \quad (5)$$

We can then define these two different implicit arc type groups as

$$FEA \subseteq (FT_{from} \times ET_{to}) , \quad (6)$$

$$EFA \subseteq (ET_{from} \times FT_{to}) , \quad (7)$$

TO (above) / (left) FROM	E_S	E_{Int}	E_E	F	P_S	P_E	AND_{EFS}	AND_{EFJ}	AND_{FES}	AND_{FEJ}	OR_{EFJ}	OR_{FES}	OR_{FEJ}	XOR_{EFJ}	XOR_{FES}	XOR_{FEJ}
E_S				→		→	→	→			→			→		
E_{Int}				→		→	→	→			→			→		
E_E																
F		→	→						→	→		→	→		→	→
P_S		→	→						→	→		→	→		→	→
P_E																
AND_{EFS}				→		→	→	→			→			→		
AND_{EFJ}				→		→	→	→			→			→		
AND_{FES}		→	→						→	→		→	→		→	→
AND_{FEJ}		→	→						→	→		→	→		→	→
OR_{EFJ}				→		→	→	→			→			→		
OR_{FES}		→	→						→	→		→	→		→	→
OR_{FEJ}		→	→						→	→		→	→		→	→
XOR_{EFJ}				→		→	→	→			→			→		
XOR_{FES}		→	→						→	→		→	→		→	→
XOR_{FEJ}		→	→						→	→		→	→		→	→

Fig. 3. Implicit arc types are a subset of the product of implicit element types. The arcs in the grey cells are Function-Event-Arcs; those arcs in the white cells are Event-Function-Arcs.

In the following, the definition of implicit arc type groups will be used in a redefinition of EPCs. The advantages of such a definition are presented in section four.

3. EPC Syntax Properties

3.1. Syntactical Constraints of Flat EPCs

Before presenting the syntactical properties of flat EPCs we still need some more definitions. Apart from cardinality which is defined in a different way by using implicit arc types, we follow Nüttgens/Rump [NR02]. The antisymmetry constraint is added, just like the constraint of the graph having to be simple which cannot be controlled by cardinality constraints on connectors.

Let E_S, E_{Int}, E_E, F, P_S, P_E, AND_{EFS}, AND_{EFJ}, AND_{FES}, AND_{FEJ}, OR_{EFJ}, OR_{FES}, OR_{FEJ}, XOR_{EFJ}, XOR_{FES}, XOR_{FEJ} be sets of elements of the respective element types. Then a *set of vertices V* is

$$V = E_S \cup E_{Int} \cup E_E \cup F \cup P_S \cup P_E \cup AND_{EFS} \cup AND_{EFJ} \cup AND_{FES} \cup AND_{FEJ} \cup OR_{EFJ} \cup OR_{FES} \cup OR_{FEJ} \cup XOR_{EFJ} \cup XOR_{FES} \cup XOR_{FEJ} \quad (8)$$

with all the elements of the union being mutually disjoint. Referring to the definitions (6) and (7) a *set of arcs A* is defined as

$$A = FEA \cup EFA. \quad (9)$$

The *precondition* of a vertex is made up by the set of ancestor arcs written as

$$\rightarrow v := \{(x,v) \in A\} \text{ with } v,x \in V . \quad (10)$$

The *postcondition* of a vertex is defined as the set of descending arcs:

$$v \rightarrow := \{(v,x) \in A\} \text{ with } v,x \in V . \quad (11)$$

A *cycle set C* is a set of vertices building a cycle:

$$C \subseteq V = \{v_1,v_2,v_3,..,v_n\} \text{ with } v_1 \rightarrow = \rightarrow v_2,\ v_2 \rightarrow = \rightarrow v_3,\ \ldots\ ,\ v_n \rightarrow = \rightarrow v_1 \quad (12)$$

Then, a flat EPC Schema $EPC_{flat} = (V,A)$ has the following *flat EPC* properties:

1. EPC_{flat} is a directed graph.
2. EPC_{flat} is a simple graph forbidding reflexive arcs or multiple arcs between two vertices.
3. EPC_{flat} is a coherent graph.
4. EPC_{flat} is an antisymmetric graph.
5. Concerning cycles: $\forall\ C_i \subseteq V: C_i \cap (E_{Int} \cup F) \neq \emptyset$.
6. The set of Events $E = E_S \cup E_{Int} \cup E_E \neq \emptyset$.
7. The set of Functions $F \neq \emptyset$.

Concerning vertices there are the following *cardinality* constraints:

1. Start vertices: $\forall v \in E_S \cup P_S$: $\rightarrow v = \varnothing$ and $|v\rightarrow| = 1$.
2. End vertices: $\forall v \in E_S \cup P_S$: $|\rightarrow v| = 1$ and $v\rightarrow = \varnothing$.
3. Inner Events: $\forall v \in E_{Int}$: $|\rightarrow v| = 1$ and $|v\rightarrow| = 1$.
4. Functions: $\forall v \in F$: $|\rightarrow v| = 1$ and $|v\rightarrow| = 1$.
5. Splits: $\forall v \in AND_{EFS} \cup AND_{FES} \cup OR_{FES} \cup XOR_{FES}$: $|\rightarrow v| = 1$ and $|v\rightarrow| > 1$.
6. Joins: $\forall v \in AND_{EFJ} \cup AND_{FEJ} \cup OR_{EFJ} \cup OR_{FEJ} \cup XOR_{EFJ} \cup XOR_{FEJ}$: $|\rightarrow v| > 1$ and $|v\rightarrow| = 1$.

Concerning vertex types the following *type consistency* constraints apply:

1. Start Events: $\forall v \in E_S$: $v\rightarrow \subseteq EFA$.
2. Inner Events: $\forall v \in E_{Int}$: $\rightarrow v \subseteq FEA$ and $v\rightarrow \subseteq EFA$.
3. End Events: $\forall v \in E_E$: $\rightarrow v \subseteq FEA$.
4. Start ProcessInterface: $\forall v \in P_S$: $v\rightarrow \subseteq FEA$.
5. End ProcessInterface: $\forall v \in P_E$: $\rightarrow v \subseteq EFA$.
6. Function: $\forall v \in F$: $\rightarrow v \subseteq EFA$ and $v\rightarrow \subseteq FEA$.
7. Event-Function-Connects: $\forall v \in AND_{EFS} \cup AND_{EFJ} \cup OR_{EFJ} \cup XOR_{EFJ}$: $\rightarrow v \in EFA$ and $v\rightarrow \in EFA$.
8. Function-Event-Connects: $\forall v \in AND_{FES} \cup AND_{FEJ} \cup OR_{FES} \cup OR_{FEJ} \cup XOR_{FES} \cup XOR_{FEJ}$: $\rightarrow v \in FEA$ and $v\rightarrow \in FEA$.

3.2. Syntactical Constraints of Hierarchical EPCs

Let $v \in V$ be a vertex, $x_i \in NC = (E_S \cup E_{Int} \cup E_E \cup F \cup P_S \cup P_E)$ and $c_i \in C = V - (E_S \cup E_{Int} \cup E_E \cup F \cup P_S \cup P_E)$ a connector, then the *non-connector ancestor border* $NCAB_v$ of v is a set of all vertices, which are transitive ancestors of v only via connectors:

$$NCAB_v = \{x_1, \ldots, x_n\}: \forall x_i \in NC: \exists n_i \in N \text{ and } c_{ij} \in C: x_i\rightarrow = \rightarrow c_{i1}, \; c_{i1}\rightarrow = \rightarrow c_{i2}, \ldots, c_{ini}\rightarrow = \rightarrow v \quad (14)$$

and the *non-connector descendant border* $NCDB_v$ of v is defined as:

$$NCDB_v = \{x_1, \ldots, x_n\}: \forall x_i \in NC: \exists n_i \in N \text{ and } c_{ij} \in C: v\rightarrow = \rightarrow c_{i1}, \; c_{i1}\rightarrow = \rightarrow c_{i2}, \ldots, c_{ini}\rightarrow = \rightarrow x_i \quad (15)$$

Let EPC_{Set} be a set of EPC Schemata

$$EPC_{Set} = \{S_1, \ldots, S_n\}, \quad (16)$$

then a hierarchical EPC Schema EPC_{hier} is defined as

$$EPC_{hier} = (V,A,H) \quad (17)$$

with H being a hierarchy relation linking a ProcessInterface or a function to another EPC Schema:

$$H \subseteq (F \cup P_E) \text{ x } EPC_{Set} \text{ with } (F \cup P_E) \subseteq V \quad (18)$$

An EPC Schema Set is an EPC_{Set} for which holds:

$$EPC_{Schema} = \{S_1, \ldots, S_n\} \quad (19)$$

fulfilling the condition:

$$\forall\ S_i \in EPC_{Schema}\text{: } S_i = (V_i,A_i,H_i) \text{ with } H_i \subseteq (F_i \cup P_{Ei}) \text{ x } EPC_{Schema}\ . \quad (20)$$

Let $HC_S \subseteq EPC_{Schema}$ then HC_S is called the hierarchical closure on S with

$$HC_S = \{S_1,\ldots,S_n \mid \forall S_i\ \exists n_i\text{: } (H_{i1} \in S \wedge Sch_{i1} \in H_{i1}) \wedge (H_{i2} \in Sch_{i1} \wedge Sch_{i2} \in H_{i2}) \wedge \ldots \wedge (H_{ini} \in Sch_{ini-1} \wedge Sch_{in} \in H_{in}) \wedge (Sch_{i1},\ldots,Sch_{ini} \in EPC_{Schema})\} \quad (21)$$

A hierarchical EPC Schema has to meet the following *hierarchy* requirements:

1. $\forall\ S_i \in EPC_{Schema}$: S_i meets the conditions for a flat EPC Schema.
2. Cardinality H_i: $\forall\ v \in P_{Ei}$: $\mid \{A \in EPC_{Schema} \mid (v,A) \in H_i\} \mid = 1$.
3. Cardinality H_i: $\forall\ v \in F_i$: $\mid \{A \in EPC_{Schema} \mid (v,A) \in H_i\} \mid \leq 1$.
4. Pre-Event-Consistency: $\forall\ v \in F_i$: $(v,S_i) \in H_i$: $NCAB_v = \{e_i \mid e_i \in E_S \wedge S_i\}$
5. Post-Event-Consistency: $\forall\ v \in F_i$: $(v,S_i) \in H_i$: $NCDB_v = \{e_i \mid e_i \in E_E \wedge S_i\}$
6. Pre-Event-Consistency: $\forall\ v \in P_{Ei}$: $(v,S_i) \in H_i$: $NCAB_v = \{e \mid e, ps \in S_i \wedge ps \in P_S \wedge e \in E_{Int} \wedge \rightarrow e = ps \rightarrow\}$. Due to our restriction on P_S $|NCAB_v| = 1$.
7. Post-Event-Consistency: $\forall\ v \in P_{Ei}$: $(v,S_i) \in H_i$: $NCDB_v = \{e \mid e, pe \in S_i \wedge pe \in P_E \wedge e \in E_{Int} \wedge e \rightarrow = \rightarrow pe\}$. Due to our restriction on P_E $|NCDB_v| = 1$.
8. Recursion prohibited: $\forall\ S_i \in EPC_{Schema}$: $S_i \notin HC_{Si}$.

3.3. Syntactical Constraints of Arcs

In section four EPC syntax checks are discussed. We will refer to the EPC syntactical constraints as *Flat 1-8* for flat EPC Schema properties, *Card 1-6* for cardinality constraints, *Type 1-8* for type consistency constraints, and *Hier 1-8* for hierarchy constraints. Here, we still need to mention constraints on arcs. Relating to our definition (2) - (5) of the different arc types, this seems redundant. But when it comes to model checking, implicit arc type groups will be a label on the respective arc which does not have to be consistent or valid. In that context there is a need to check if the types of the referenced vertices match the implicit arc type. Thus, we add *Arc 1-2* to check *arc consistency*:

1. $\forall\ (x,y) \in FEA$: $x \in FT_{from}$ and $y \in ET_{to}$.
2. $\forall\ (x,y) \in EFA$: $x \in ET_{from}$ and $y \in FT_{to}$.

4. Using Implicit Arc Types for Validity Checks

4.1. Problems without Implicit Types

Type Consistency of connectors has been a problem of former EPC definitions without implicit elements and arc types. When there is a path of successive connectors as in Fig. 4, transitive non-connector ancestor border $NCAB_v$ and descendant border $NCDB_v$ have to be determined. This involves costly traversing of the EPC graph. In the following paragraph, we describe how implicit element and arc types can lead to a better performance. In that context we distinguish guided modelling and model checking.

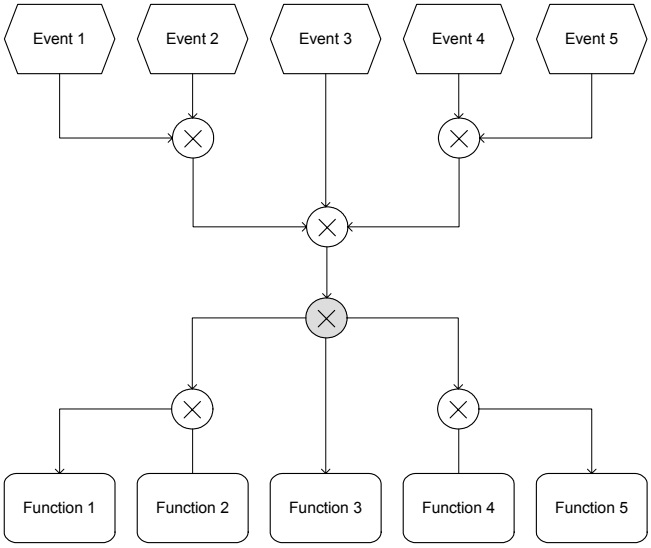

Fig. 4. In order to check Type Consistency of the grey XOR-connector the transitive non-connector ancestors and transitive non-connector descendants have to be determined.

4.2. Guided Modelling using Implicit Types

Guided Modelling describes strategies to support the modeller in the process of building models in order to grant syntactically valid models. The definitions 6 and 7 concerning the two different implicit arc type groups can be used to determine conflicts when there is a new arc added to the model. It is assumed that the modeller uses symbols corresponding to explicit EPC element types. In order to take advantage of implicit types, we need a set of possible implicit element or arc types, referred to as $\prod$, attached to every instance of an explicit EPC symbol. Modelling includes four elementary operations: the insertion of an explicit element and the insertion of an explicit control flow arc to the model; and the deletion of an element or an arc. Changes can be interpreted as a sequence of a deletion and an addition. In the following, we concentrate on insertions, because deletions work similar in the opposite way.

When there is an element inserted into the model, $\prod$ is instantiated with all of its implicit element types. The insertion of an arc affects the sets of possible implicit types of the start vertex of the arc $\prod_S$; of the end vertex of the arc $\prod_E$; and of the arc itself $\prod_{arc}$. In a first step, $\prod_{arc}$ is determined by comparing $\prod_E$ and $\prod_S$ with the definitions of the two implicit arc type groups:

$$\prod_E \cap ET_{from} \neq \varnothing \ \wedge \ \prod_S \cap FT_{to} \neq \varnothing \ \Rightarrow \ EFA \in \prod_{arc}. \qquad (22)$$

$$\prod_E \cap FT_{from} \neq \varnothing \ \wedge \ \prod_S \cap ET_{to} \neq \varnothing \ \Rightarrow \ FEA \in \prod_{arc}.$$

If there is $\prod_{Arc} = \{\}$ after this first steps, the new arc is not valid, because type consistency is no longer granted. This invalid arc should then be deleted and a report be provided for the modeller. If $\prod_{Arc} \neq \{\}$, then $\prod_E$ and $\prod_S$ need to be updated as a second step according to these rules:

$$\text{If } \prod_{Arc} = \{EFA, FEA\}: \prod_E{}' = \prod_E \wedge \prod_S{}' = \prod_S \qquad (23)$$

$$\text{If } \prod_{Arc} = \{EFA\}: \prod_E{}' = \prod_E \cap ET_{from} \ \wedge \ \prod_S{}' = \prod_S \cap FT_{to}$$

$$\text{If } \prod_{Arc} = \{FEA\}: \prod_E{}' = \prod_E \cap FT_{from} \ \wedge \ \prod_S{}' = \prod_S \cap ET_{to}$$

$$\text{If } \prod_{Arc} = \{\}: \prod_E{}' = \prod_E \wedge \prod_S{}' = \prod_S.$$

As a third step, these recalculations must cascade from the updated vertices, because inconsistencies may appear transitively via connectors. Cycles do not pose a problem for termination, because $\prod$ is a finite set of maximum 16 elements. Each step can only involve a reduction of elements leading at least to a termination in terms of an empty set. This three-step algorithm helps to transform an EPC model from one type consistent state to another type consistent state. The operations from definition 22 and 23 may therefore be considered as an EPC model transaction granting validity.

4.3. Model Checking using Implicit Types

The case of guided modelling demands extra features to be added to the business process modelling tool. The model checking approach does not require such capabilities. It takes a model composed of explicit element and arc type symbols as an input and determines the implicit type for each symbol. The use of implicit types avoids the calculation of the non-connector ancestor border $NCAB_v$ and descendant border $NCDB_v$ for each connector.

The algorithm takes a list of symbols as an input, with each symbol being of one of the explicit element or arc types. The order of the symbols is arbitrary. It works in two steps. The first step is the hypothesis step: by checking the cardinality of the ancestors and descendants, the implicit types of Functions, Events and ProcessInterfaces are determined, and join connectors are distinguished from split connectors. If an unexpected constellation appears the implicit type is set to “invalid”. For arcs and connectors, we do not calculate the non-connector ancestor border, but follow only one path until we reach a non-connector element. If this is an Event, the arc is set to the implicit type EFA and

the connector to a EventFunctionJoin or –Split. This hypothesis generated by only looking at one (transitive) ancestor can have two consequences: firstly, the hypothesis is correct, or the there is a type inconsistency. It is not possible that another hypothesis is correct.

The second step is the confirmation step. The generated list of symbols labelled with implicit types is taken as an input. Now, for all elements and arcs type consistency can be checked only by looking at the ancestor and descendant arc for implicit element types or the start and the end vertex for implicit arc types. An expansion of vertices is no longer needed.

5. Conclusion and Future Work

This paper has addressed the validity of EPC business process models. As process modelling becomes more and more decentralised in organisations, non-professional modellers need tools to assist them to produce high quality models in terms of syntactical validity. The concept of implicit element and arc types gives a transparent answer to the question of valid EPC syntax. Validity of connectors can be check only by looking at the implicit arc type group of the ancestor and descendant arcs. Additionally, the implicit complexity of EPCs as a modelling technique is revealed. Nevertheless, pre- and post-event-consistency and the recursion prohibition (*Hier 4-8*) of hierarchical EPC Schemas still need closures to be calculated in order to check validity. The next step will be an XSLT based syntax check for models stored in EPC Markup Language (EPML) [MN02].

References

[ADK02] van der Aalst, W.; Desel, J.; Kindler, E.: On the semantics of EPCs: A vicious circle, in: Nüttgens, M.; Rump, F.J. (eds.): Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten - EPK 2002, Proceedings of the GI-Workshop EPK 2002, Trier, 2002, pp. 71-79.

[Aa99] van der Aalst, W.M.P.: Formalization and Verification of Event-driven Process Chains, in: Information and Software Technology 41(1999)10, pp. 639-650.

[CS94] Chen, R.; Scheer, A.-W.: Modellierung von Prozessketten mittels Petri-Netz-Theorie, in: Scheer, A.-W. (ed.): Publications of the Institut für Wirtschaftsinformatik, No. 107, Saarbrücken 1994.

[De02] Dehnert, J.: Making EPC´s fit for Workflow Management, in: Nüttgens, M.; Rump, F.J. (eds.): Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten - EPK 2002, Proceedings of the GI-Workshop EPK 2002, Trier, 2002, pp. 51-69.

[Ke99] Keller, G. & Partner: SAP/ R3 prozeßorientiert anwenden. Iteratives Prozeß-Prototyping mit Ereignisgesteuerten Prozeßketten und Knowledge Maps, Bonn et al. 1999.

[KM94] Keller, G.; Meinhardt, S.: DV-gestützte Beratung bei der SAP-Softwareeinführung, in: HMD 31(1994)175, pp. 74-88.

[KNS92] Keller, G.; Nüttgens, M.; Scheer, A.-W.: Semantische Prozeßmodellierung auf der Grundlage „Ereignisgesteuerter Prozeßketten (EPK)“. In: Scheer, A.-W. (Hrsg.): Veröffentlichungen des Instituts für Wirtschaftsinformatik, Heft 89, Saarbrücken, 1992.

[LSW98] Langner, P.; Schneider, C.; Wehler, J.: Petri Net Based Certification of Event driven Process Chains, in: Desel, J.; Silva, M. (eds.): Application and Theory of Petri Nets 1998, LNCS Vol. 1420, Springer, Berlin et. al. 1998, pp. 286-305.

[Me03] Mendling, J.: Event-Driven-Process-Chain-Markup-Language (EPML): Anforderungen, Konzeption und Anwendung eines XML-Schemas für Ereignisgesteuerte Prozessketten (EPK), in: Höpfner, H.; Saake, G.: Proceedings of the Student Program of the 10th Conference “Database Systems for Business, Technology and Web”, GI Section Databases and Information Systems, Leipzig, 25.02.2003, Madgeburg, 2003, pp. 48-50.

[MN02] Mendling, J.; Nüttgens, N.: Event-Driven-Process-Chain-Markup-Language (EPML): Anforderungen zur Definition eines XML-Schemas für Ereignisgesteuerte Prozessketten (EPK), Proceedings of the GI-Workshop EPK 2002, Trier, 2002, pp. 87-93.

[NR02] Nüttgens, M.; Rump, F.: Syntax und Semantik Ereignisgesteuerter Prozessketten (EPK), in: Prozessorientierte Methoden und Werkzeuge für die Entwicklung von Informationssystemen (Promise’2002), Hasso-Plattner-Institut für Softwaresystemtechnik an der Universität Potsdam, 9.-11. Oktober 2002, Potsdam 2002.

[Ri00] Rittgen, P.: Paving the Road to Business Process Automation, European Conference on Information Systems (ECIS) 2000, Vienna, Austria, July 3 - 5, 2000, pp. 313-319.

[Ro97] Rodenhagen, J.: Darstellung ereignisgesteuerter Prozeßketten (EPK) mit Hilfe von Petrinetzen, Diplomarbeit Universität Hamburg Fachbereich Informatik (Prof. Valk), Hamburg 1997.

[Ru99] Rump, F.: Geschäftsprozeßmanagement auf der Basis ereignisgesteuerter Prozeßketten - Formalisierung, Analyse und Ausführung von EPKs, Teubner, Stuttgart et al. 1999.

[Sc00] Scheer, A.-W.: ARIS – Business Process Modeling, 3rd edition, Springer, Berlin et. al. 2000.