Markus Nüttgens, Frank J. Rump (Hrsg.)

EPK 2003

Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten

2. Workshop der Gesellschaft für Informatik e.V. (GI)
und Treffen ihres Arbeitkreises „Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten (WI-EPK)“

08. Oktober 2003 in Bamberg

Proceedings

Veranstalter

veranstaltet vom GI-Arbeitskreis "Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten (WI-EPK)" der GI-Fachgruppe WI-MobIS (FB-WI) in Kopperation mit der GI-Fachgruppe EMISA (FB-DBIS) und der GI-Fachgruppe Petrinetze (FB-GInf).

Dr. Markus Nüttgens (Sprecher)
Email: markus@nuettgens.de

Prof. Dr. Frank J. Rump (stellv. Sprecher)
Email: rump@informatik-emden.de

EPK 2003 / Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten. Hrsg.: Markus Nüttgens, Frank J. Rump. – Bamberg 2003

EPC Syntax Validation with XML Schema Languages

Jan Mendling

Universität Trier
E-Mail: mendling@web.de

Markus Nüttgens

Universität des Saarlandes
E-Mail: markus@nuettgens.de

Abstract: This paper addresses syntactical validation of Event-Driven Process Chains (EPC), in particular the syntactical validation of consecutive connectors. Building on the distinction between implicit and explicit element and arc types a definition of EPC syntax properties is given. Different XML Schema languages are examined in how far they are able to express this definition. The contribution of this paper is twofold: it is shown how EPCs can be represented in an XML syntax, called EPC Markup Language (EPML), and it is demonstrated how Schematron can be used to implement the EPC syntax definition as an EPML Schema validator.

1. Introduction

Event Driven Process Chains (EPC) are a popular technique for model business processes on a conceptual level [KNS92]. In order to leverage the benefits of business process modeling, process documentations should be reused in workflow applications or exchanged between different business process modeling tools. This avoids time-consuming re-input of data, which is less error prone and which grants consistency between models in different applications. Even for a small number of tools involved, it becomes efficient to define an intermediary or exchange format.

The approach towards an EPC Markup Language (EPML) [MN02] follows the guiding principles of readability, extensibility, tool orientation, and syntactical correctness [MN03b]. The first three features mainly address the presence of certain markup elements. The fourth aspect deals with correctness rules concerning the logical sequence of different EPC elements.

There have been a lot of the contributions on EPCs focusing 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].

Type consistency of consecutive connectors poses a major problem for these definitions. Fig. 1 illustrated this problem: when there is a chain of consecutive connectors, the syntactical correctness of them depends on the types of transitive non-connector ancestors and descendants. The connectors in the example are both correct because their transitive ancestors are all Events and the transitive descendants are all Functions. Mendling/Nüttgens [MN03a] address this problem by introducing implicit types for elements and arcs. Such a concept allows validation of type consistency of connector chains without taking transitive ancestors and descendants into consideration. Section two will present a respective definition of flat EPCs including EPC syntax rules that allow one to determine type consistency by checking only direct ancestors and descendants. Properties of hierarchical EPCs can be found in [NR02]. Section three will examine in how far different XML Schema Languages are capable to express these flat EPC syntax rules. Finally, we will show how Schematron, a rule-based XML Schema language, can be used to perform EPC syntax checks for process models expressed in EPC Markup Language (EPML), an XML syntax for EPCs.

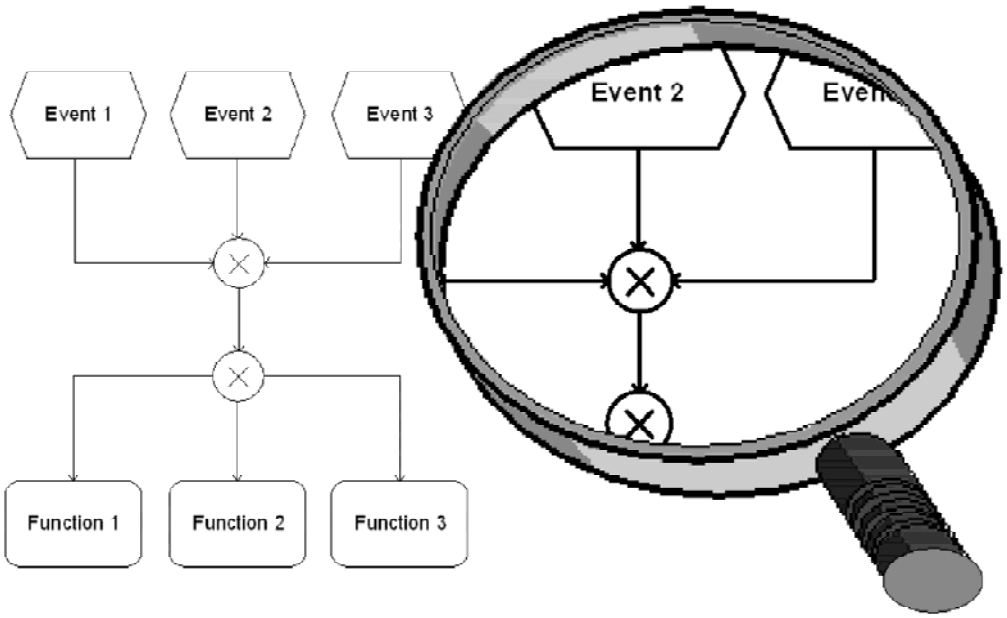

Fig. 1. Looking through the magnifier illustrates that syntactical correctness of the first connector cannot be determined by checking only direct ancestors and descendants.

2. EPC Syntax Properties and Implicit Types

2.1. Explicit and Implicit Element Types

The distinction of explicit and implicit element types is rooted in two perspectives on business process modelling, the perspective of the modeller and 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 [Si97]. This set of symbols is provided by a business process modelling tool. 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.

Implicit element types relate to the perspective of syntactical verification. Each implicit element type captures a specific constellation in which explicit element types may occur implying different restrictions on the set of allowed ancestors and descendants, and their cardinality [MN03a]. Events E may be Start Events E_S, Inner Event E_{Int} or End Event E_E. 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].

2.2. Explicit and Implicit Arc Types

Analogously to the distinction between explicit and implicit element types, implicit arc types are defined 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. 2 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.

$$\text{Event Types (from) } ET_{from} := E_S \cup E_{Int} \cup AND_{EFS} \cup AND_{EFJ} \cup OR_{EFJ} \cup XOR_{EFJ} \quad (1)$$

$$\text{Function Types (from) } FT_{from} := F \cup P_S \cup AND_{FES} \cup AND_{FEJ} \cup OR_{FES} \cup OR_{FEJ} \cup XOR_{FES} \cup XOR_{FEJ}\,, \quad (2)$$

$$\text{Event Types (to) } ET_{to} := E_{Int} \cup E_E \cup AND_{EFS} \cup AND_{EFJ} \cup OR_{EFJ} \cup XOR_{EFJ}\,, \quad (3)$$

$$\text{Function Types (to) } FT_{to} := F \cup P_E \cup AND_{EFS} \cup AND_{EFJ} \cup OR_{EFJ} \cup XOR_{EFJ}\,. \quad (4)$$

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. 2. 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.

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

$$FEA \subseteq (FT_{from} \times ET_{to}) , \quad (5)$$

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

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.

2.3. Syntactical Constraints of Flat EPCs

Before presenting the syntactical properties of flat EPCs we still need some more definitions. We restrict our discussion to flat EPC properties as presented in [MN03a], which are based on the definitions in [NR02]. 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

$$V := E_S \cup E_{Int} \cup E_E \cup F \cup P_S \cup P_E \cup AND_{EFS} \cup AND_{EFJ} \cup AND_{FES} \cup AND_{FEJ} \cup OR_{EFJ} \cup OR_{FES} \cup OR_{FEJ} \cup XOR_{EFJ} \cup XOR_{FES} \cup XOR_{FEJ} \quad (7)$$

with all the elements of the union being mutually disjoint. Referring to the definitions (5) and (6) a *set of arcs A* is defined as

$$A := FEA \cup EFA. \quad (8)$$

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

$$\rightarrow v := \{a \in A \mid a = (x,v) \wedge x,v \in V\} . \quad (9)$$

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

$$v\rightarrow := \{a \in A \mid a = (v,y) \wedge v,y \in V\} . \quad (10)$$

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

$$C \subseteq V = \{v_1,v_2,v_3,..,v_n\} \text{ with } v_1\rightarrow = \rightarrow v_2,\ v_2\rightarrow = \rightarrow v_3,\ \ldots\ ,\ v_n\rightarrow = \rightarrow v_1 \quad (11)$$

Then, a flat EPC Schema EPC_{flat} 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. Cycles made up only of connectors are forbidden:
   $\forall C \subseteq V: C \cap (E_{Int} \cup F) \neq \emptyset$.
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: $\forall v \in E_S \cup P_S: \rightarrow v = \emptyset$ and $|v\rightarrow| = 1$.
2. End vertices: $\forall v \in E_S \cup P_S: |\rightarrow v| = 1$ and $v\rightarrow = \emptyset$.
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 $v\rightarrow \in FEA$.

In section four EPC syntax checks are discussed. We will refer to the EPC syntactical constraints as *Flat 1-7* for flat EPC Schema properties, *Card 1-6* for cardinality constraints, and *Type 1-8* for type consistency constraints. Here, we still need to mention constraints on arcs. Relating to our definition (1) - (4) 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 correct. 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 $y \in FT_{to}$.

3. EPC Markup Language (EPML) and XML Schema Languages

3.1. EPML by Example

Fig. 3 shows two EPC processes that both conform to the EPC syntax requirements. The Start Event is followed by a Function "List Requirement". Via a XOR connector two Events may occur. The "Design Process" is linked to "Can be fulfilled" by a ProcessInterface. The second process starts with the same ProcessInterface and the same Event "Can be fulfilled" follows. Afterwards we present EPML code for these two processes. It is simplified in that sense that it only describes the control flow, tool-orientation and extensibility issues are left out.

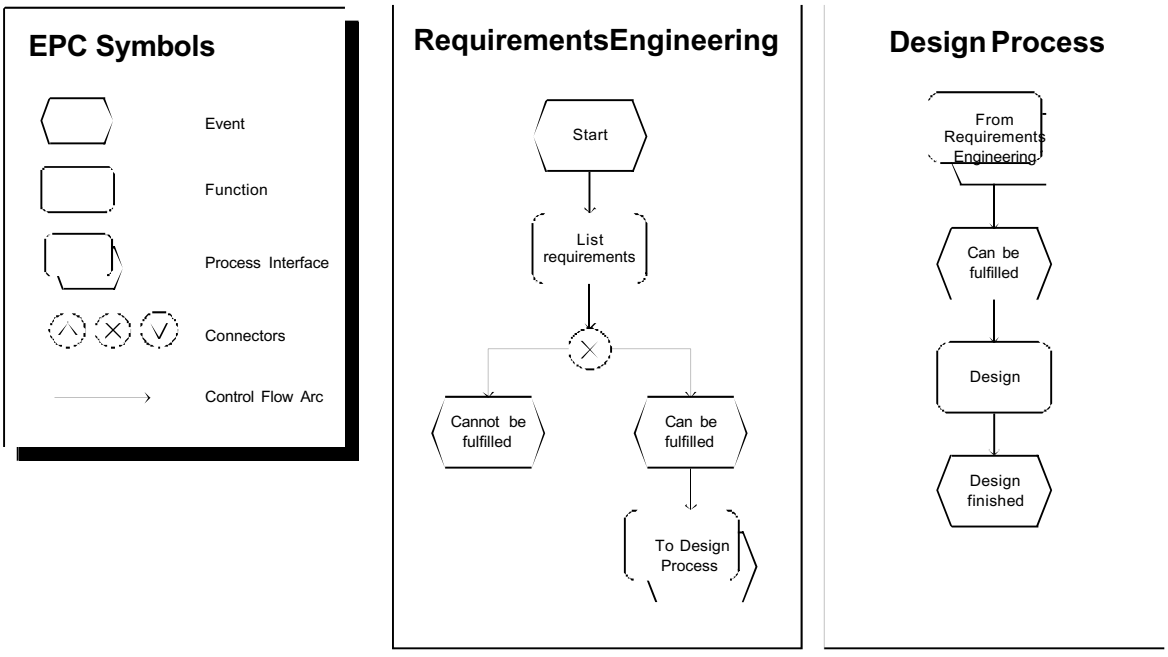

Fig. 3. Example of two flat EPC processes.

The code snippet in fig. 4 shows the corresponding representation in EPML according to [MN03b]. In the <Definitions> part objects are defined in order to allow their reusability via multiple references in one or more <EPC> processes. Connectors are excluded from this. The DefId attribute is set to a number unique in the context of <Definitions>, and later referenced as DefRef in process objects. The name attribute allows a description of the object. The <EPC> part is identified by a unique EpcId attribute, a name shall also be allowed. The explicit type is taken for markup, the implicit type is featured in the type attribute. <Arc> tags are identified by a unique combination of FromId and ToId attributes. They also have a type attribute indicating their implicit arc type.

```
<EPML>
    <Definitions>
       <EventDef DefId="1" name="Start"/>
       <FunctionDef DefId="2" name="List Requirements"/>
       <EventDef DefId="3" name="Cannot be fulfilled"/>
       <EventDef DefId="4" name="Can be fulfilled"/>
       <ProcessInterfaceDef DefId="5" name="Design Process"/>
       <FunctionDef DefId="6" name="Design"/>
       <EventDef DefId="7" name="Design finished"/>
    </Definitions>
    <EPC EpcId="1" name="Requirements Engineering">
       <Event DefRef="1" Id="11" type="EventStart"/>
       <Function DefRef="2" Id="12" type="Function">
       <Arc FromId="11" ToId="12" type="EventFunctionArc"/>
       <XOR Id="13" type="XORFunctionEventSplit"/>
       <Arc FromId="12" ToId=13" type="FunctionEventArc"/>
       <Event DefRef="3" Id="14" type="EventEnd"/>
       <Arc FromId="13" ToId="14" type="FunctionEventArc"/>
       <Event DefRef="4" Id="15" type="EventInternal"/>
       <Arc FromId="13" ToId="15" type="FunctionEventArc"/>
       <ProcessInterface DefRef="5" Id="16" type="ProcessIntEnd">
              <ToProcess EpcLinkId="2"/>
       </ProcessInterface>
       <Arc FromId="15" ToId="16" type="EventFunctionArc"/>
    </EPC>
    <EPC EpcId="2" name="Design Process">
       <ProcessInterface DefRef="5" Id="21" type="ProcessIntStart"/>
       <Event DefRef="4" Id="22" type="EventInternal"/>
       <Arc FromId="21" ToId="22" type="FunctionEventArc"/>
       <Function DefRef="6" Id="23" type="Function"/>
       <Arc FromId="22" ToId="23" type="EventFunctionArc"/>
       <Event DefRef="7" Id="24" type="EventEnd"/>
       <Arc FromId="23" ToId="24" type="FunctionEventArc"/>
    </EPC>
</EPML>
```

Fig. 4. Example of an EPML representation of the two flat EPC processes given in Fig. 3.

3.2. XML Schema Languages

In the following we will examine different XML schema languages. They are able to express the structure given by the example above, but they differ in their ability to express EPC syntax properties which are presented in section two. XML schema languages play an important role when it comes to exactly describing a class of XML documents. Document Type Definition (DTD) [Br00] has been the first schema language for XML, granting continuity from SGML where DTDs have also been used.

But with the spread of XML, new schema languages have been developed. Mainly, these efforts have been motivated by shortcomings of DTDs: lack of namespace support, non-XML syntax, and a lack of a mechanism to express constraints [Og00]. We will follow van der Vlist [Vl02] to distinguish three major groups: object-oriented XML schema languages, grammar-based XML schema languages, and rule-based XML schema languages. W3C XML Schema will be discussed as an object-oriented language, RELAX NG as a grammar-based approach, and Schematron as a rule-based schema language.

3.3. W3C XML Schema

W3C XML Schema [BM01], [Be01] is a schema language introduced by the World Wide Web Consortium to overcome shortcomings of DTDs. A major advantage is the much higher flexibility in defining uniqueness constraints. The old ID and IDREF data types of DTDs can still be used, but <xs:unique>, <xs:key>, and <xs:keyref> enable uniqueness constraints on arbitrary data types, in definable contexts, and for both attribute and element nodes. Nevertheless, W3C XML Schema has three grave disadvantages. Firstly, constraints can only be expressed in terms of restrictive content models or by using the above described uniqueness constructs. Take the four constraint packages defined in section 2. Flat 1 of EPCs being a directed graph can be handled by modelling arcs to have a FromId and a ToId. Flat 2 demanding for a simple graph can be controlled by forbidding multiple arcs between two vertices with a uniqueness constraint, but reflexive arcs cannot be prohibited by a uniqueness constraint. Concerning Flat 3-5 coherence, antisymmetry, and cycles also cannot be expressed with the help of uniqueness constraints.

For the objects within <EPC> a <xs:choice maxOccurs="unbounded"> is the appropriate definition to allow arbitrary sequences of any kind of object. To meet the existence-constraints of a minimum of one Event and one Function in an EPC of *Flat 6-7* there should be one `<Function>` and one `<Event>` defined before that choice. This is still okay with W3C XML Schema, but we have fixed the place of one mandatory Event and Function. An idea might be to add another leading choice block similar to the other. But this is forbidden according to the Unique Particle Attribution Rule, already a part of DTDs as nondeterministic content models. This is the second major disadvantage of W3C XML Schema. Consider the schema validator to find an `<Event>`. It would not know whether this element belongs to the choice definition or the element declaration. In formal languages this is not a problem, but in W3C XML Schema it is forbidden.

In general, the cardinality constraints *Card 1-6* could be expressed by a uniqueness constraint as a cardinality of one ancestor or one descendant respectively is demanded. For example, the `FromId` of a ProcessInterface-Event-Arc would have to be unique because there is only one arc from a ProcessInterface allowed. Unfortunately, this constraint would have to be declared in the `<arc>` container fixing it for all arc types. Taking an XML structure as presented above will not allow one to control cardinality via W3C XML Schema uniqueness checks.

The *Type 1-8* constraints pose a problem due to the third shortcoming of W3C XML Schema, the so called Consistent Declaration Rule. The `Ids` of objects could be modelled as belonging to a specific simple type, i.e. by defining a prefix using a regular expression ("Event[0-9]*"). This would cause problems with the type attribute in `<Arc>`, because the Consistent Declaration Rule forbids the choice between elements of the same name and different types. The `Ids` would have to be of different types which is not allowed. Finally, *Arc 1-4* pose the same problems as *Type 1-8*.

3.4. RELAX NG

Let's now take a look at RELAX NG [CM01]. This XML schema language is based on the formal concept of regular tree grammar. Its expressive power goes beyond W3C XML Schema [MLM00]. It does not have something like a Unique Particle Attribution Rule, and is therefore able to handle nondeterministic content models. But when it comes to constraints, RELAX NG depends on external data types. This means that only DTD classic ID, IDREF and IDREFS can be used. For a readable EPML structure as presented above, this is a major shortcoming. Take the cardinality constraints *Card 1-6*. We need to define the Id's of the object types as data type ID, FromId and ToId of the arc elements as IDREF type. This forbids the control of uniqueness on the FromId or the ToId attribute, because they need to be referenced and therefore have to have the IDREF data type, which cannot be additionally declared to be a unique ID as well. This shortcoming impedes the detection of multiple arcs between two vertices via a uniqueness check, because attributes in arcs need to be of IDREF type. Thus, concerning *Flat 1-5* RELAX NG shows this disadvantage in contrast to W3C XML Schema. But thanks to its ability to express nondeterministic content models, we can use the technique described in W3C XML Schema section to grant the existence of at least one Event and one Function element. If we consider the type constraints of ancestor and descendant vertices *Type 1-8* and also *Arc 1-2*, we face the same problems as already with W3C XML Schema. Even though RELAX NG is more powerful to describe structure than W3C XML Schema, it is less flexible to check uniqueness, which is a disadvantage in the context of our document structure. We will now have a look at Schematron belonging to the family of rule-based schema languages.

3.5. Schematron

Schematron [Je02] follows a different paradigm than W3C XML Schema and RELAX NG. Instead of declaring permitted structures Schematron checks for structures that are not permitted. For this purpose Schematron allows assertions to be declared which state positive properties of an instance document formulated as an XPath [CD99] expression. This allows a great flexibility due to XPath built-in functions like element count or string manipulation. A wide range of constraints can be expressed which is not possible using W3C XML Schema or RELAX NG.

As *Flat 1* is already granted by the structure of the arc elements, we continue with *Flat 2* demanding the EPC graph to be simple. Consider the following code which declares a rule in the context of <Arc> elements:

```
<sch:rule context="Arc">
    <sch:assert test="./@FromId!=./@ToId">An arc is not reflexive
    </sch:assert>
</sch:rule>
```

This check for reflexivity is easy to implement with Schematron, just like the check for multiple arcs and coherence. The constraint for cycles demand closure calculation, they cannot be formulated as XPath expressions. *Flat 6-7* concerning the existence of Events and Functions can easily be asserted with Schematron.

```
<sch:rule context="EPC">
    <sch:assert test="count(./Event)>1"/>
</sch:rule>
```

The cardinality constraints *Card 1-6* are also easy to specify with Schematron, for example take the Function element:

```
<sch:rule context="Function">
    <sch:assert test="count(./@Id=../Arc/@FromId)=1"/>
    <sch:assert test="count(./@Id=../Arc/@ToId)=1"/>
</sch:rule>
```

The type consistency constraints *Type 1-8* can also be expressed using XPath. E.g. *Type 7* imposes type restrictions for Event-Function-Connects:

```
<sch:rule context="AND|OR|XOR">
    <sch:assert test="
       (type='ANDEventFunctionSplit' or type='ANDEventFunctionJoin' or
        type='OREventFunctionJoin' or type='XOREventFuntionJoin')
        and @Id[@Id=@FromId[../@type='EventFunctionArc']]"/>
</sch:rule>
```

Similar to *Type 1-8* also *Arc 1-2* can be checked using Schematron.

4. Conclusion on the Usefulness of Implicit Element and Arc Types

Table 3 summarizes the results. Schematron provides the best mechanism to express syntactical constraints demanded by the formal definition of EPCs. W3C XML Schema shows some advantages over RELAX NG when it comes to uniqueness assertions. But both of them do not permit the expression of constraints as it is possible in Schematron. Only coherence and cycles (*Flat 3*, *Flat 5*) cannot be checked even with Schematron, because the EPC graph has to be traversed for these properties.

The good performance of Schematron is mainly a result of the EPC definition using implicit element and arc types. Implicit types avoid graph expansion and closure calculation for type consistency checks of connectors. Therefore, this EPC syntax definition has proved valuable in conjunction with the definition of an EPC Markup Language (EPML). It allows a straight forward EPC syntax validation (except *Flat 3* and *Flat 5*) given an EPML file as input, a Schematron definition of EPML, and a Schematron schema validator. This represents a major simplification of EPC syntax validation.

	W3C XML Schema	RELAX NG	Schematron
Flat 1: Directed	via structure	via structure	via structure
Flat 2: Simple	partly via uniqueness	no	via rule
Flat 3: Coherent	no	no	no
Flat 4: Antisymmetric	no	no	via rule
Flat 5: Cycles	no	no	no
Flat 6: ∃ Event	partly via structure	via structure	via rule
Flat 7: ∃ Function	partly via structure	via structure	via rule
Card 1: Start	no	no	via rule
Card 2: End	no	no	via rule
Card 3: Int. Event	no	no	via rule
Card 4: Functions	no	no	via rule
Card 5: Splits	no	no	via rule
Card 6: Joins	no	no	via rule
Type 1: Start Event	no	no	via rule
Type 2: Int. Event	no	no	via rule
Type 3: End Event	no	no	via rule
Type 4: Start Proc.-I.	no	no	via rule
Type 5: End Proc.-I.	no	no	via rule
Type 6: Function	no	no	via rule
Type 7: EF-connects	no	no	via rule
Type 8: FE-connects	no	no	via rule
Arc 1: FEA	no	no	via rule
Arc 2: EFA	no	no	via rule

Table 1. The summary of the findings: XML schema languages and their ability to express the constraints necessary to decide syntactical correctness of EPCs represented in EPML.

References

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

[ADK02] v.d. Aalst, W.M.P.; 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

[Be01] Beech, D.; Lawrence, S.; Moloney, M.; Mendelsohn, N.; Thompson, H.S. (eds.): XML Schema Part 1: Structures. World Wide Web Consortium, USA, 2001.

[BM01] Biron, P.V.; Malhotra, A. (eds.): XML Schema Part 2: Datatypes. World Wide Web Consortium, USA, 2001.

[Br00] Bray, T. et al. (eds.): Extensible Markup Language (XML) 1.0 (Second Edition). World Wide Web Consortium, USA, 2000.

[CD99] Clark, J.; DeRose, S.: XML Path Language (XPath) Version 1.0, World Wide Web Consortium, Boston, USA, 1999.

[CM01] Clark, J.; Murata, M.: RELAX NG Specification, 3 December 2001. URL: http://www.relaxng.org/spec-20011203.html.

[CS94] Chen, R.; Scheer, A.-W.: Modellierung von Prozessketten mittels Petri-Netz-Theorie, in: Scheer, A.-W. (ed.): Veröffentlichungen des Instituts für Wirtschaftsinformatik, Heft 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.

[Je02] Jelliffe, R.: The Schematron Assertion Language 1.5, 2002-10-01. URL: http://www.ascc.net/xml/resource/schematron/Schematron2000.html.

[KNS92] Keller, G.; Nüttgens, M.; Scheer, A.-W.: Semantische Prozeßmodellierung auf der Grundlage „Ereignisgesteuerter Prozeßketten (EPK)". In: Scheer, A.-W. (ed.): 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. (eds.): 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.

[MLM00] Murata, M.; Lee, D.; Mani, M.: Taxonomy of XML Schema Languages using Formal Language Theory. In: Extreme Markup Languages, Montreal 2001. URL: http://www.cobase.cs.ucla.edu/tech-docs/dongwon/mura0619.pdf.

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

[MN03a] Mendling, J.; Nüttgens, M.: EPC Modelling Based on Implicit Arc Types. In: Godlevsky, M.; Liddle, S.W.; Mayr, H.C. (eds.): Information Systems Technology and its Applications, International Conference ISTA'2003, June 19-21, 2003, Kharkiv, Ukraine, Proceedings. LNI 30, pp. 131-142.

[MN03b] Mendling, J.; Nüttgens, M.: XML-basierte Geschäftsprozessmodellierung. In: Uhr, W., Esswein, W.; Schoop, E. (eds): Wirtschaftsinformatik 2003 / Band II, Heidelberg, 2003, pp. 161 -180.

[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.

[Og00] Ogbuji, C.: Validating XML with Schematron, O'Reilly XML.com, Sebastopol et. al. 2000. URL: http://www.xml.com/pub/a/2000/11/22/schematron.html.

[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.

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

[St73] Sinz, E.: Modell. In: Mertens, P. et al. (eds.): Lexikon der Wirtschaftsinformatik, Berlin et al., 1997, S. 270.

[Vl02] van der Vlist, E.: XML Schema, O'Reilly, Sebastopol et. al. 2002.