Technical Report JM-2005-03-10
Vienna University of Economics and Business Administration

EPC Markup Language (EPML)

An XML-Based Interchange Format for Event-Driven Process Chains (EPC)

Jan Mendling[1], **Markus Nüttgens**[2]

[1] Department of Information Systems and New Media, Vienna University of Economics and Business Administration, A-1180 Vienna, Austria
(e-mail: jan.mendling@wu-wien.ac.at)
[2] Chair of Information Systems, HWP Hamburg, D-20146 Hamburg, Germany
(e-mail: nuettgens@hwp-hamburg.de)

Key words Business Process Management, Interchange Formats, Event-Driven Process Chains (EPC), XML, EPML

Version 2005-03-10

Abstract This article presents an XML-based interchange format for Event-Driven Process Chains (EPC) that is called EPC Markup Language (EPML). EPML builds on EPC syntax related work and is tailored to be a serialization format for EPC modelling tools. Design principles inspired by other standardization efforts and XML design guidelines have governed the specification of EPML. After giving an overview of EPML concepts we present examples to illustrate its features including flat and hierarchical EPCs, business views, graphical information, and syntactical correctness.

1 Introduction

Today business process modelling is mainly used in two different contexts: business analysts use process models for documentation purposes, process optimization and simulation; information system analysts use them on the middleware tier in order to glue together heterogeneous systems. For both kinds of application analysts can choose from a variety of tools in order to support modelling of processes. Gartner Research (2002) distinguishes 35 major vendors of such software. Heterogeneity of concepts and data formats cause interoperability problems between these tools. A survey of Delphi Group (2003) identifies the lack of a common and accepted interchange format for business process models as the major encumbrance to business process management.

Event-Driven Process Chains (EPC) introduced by Keller et al. (1992) are a wide-spread method for business process modelling. Keller and Teufel (1998) describe the application of EPCs in the context of the SAP reference model. Motivated by the heterogeneity of business process modelling tools, a proposal for an interchange format for EPCs has been developed by Mendling and Nüttgens (2002, 2003b,c). It is called EPC Markup Language (EPML). In the domain of business process modelling the establishment of a standardized representation may be even more beneficial than in other areas, because it can be used in two different directions: *horizontal interchange* simplifies the integration of BPM tools of the same scope. *Vertical interchange* leverages the integration of simulation engines, execution engines, and monitoring engines as intended by interface definitions of the WfMC (2002). Standardization might be a crucial step to close the engineering gap between business process modelling and implementation.

This article gives an overview of EPML. Section 2 introduces Event-Driven Process Chains (EPCs) as a method for business process modelling, their syntactical elements, and related research on EPC syntax. Section 3 discusses EPML general design principles and XML design guidelines that have guided the specification. Section 4 explains the structure of EPML and how the different syntax elements relate to each other. Furthermore, it is outlined why edge element lists are used to describe EPC process graphs in EPML. Section 5 introduces specific aspects of EPML illustrated by examples including flat EPCs, hierarchies of EPCs, business perspectives, graphical representation, and syntactical correctness. Section 6 concludes the paper and gives an outlook on future work.

2 Event-Driven Process Chains (EPCs)

In Keller et al. (1992) the EPC is introduced as a modelling concept 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). The four resulting combinations are discussed for each of the three connectors yielding twelve possibilities. OR-Split and XOR-Split are prohibited subsequent to events. This restriction refers to the semantics of events as passive elements which are unable to determine the functions that should follow. Based on practical experience with the SAP Reference model, Keller and Meinhardt (1994) introduce process interfaces and hierarchical functions as additional element types of EPCs. These two elements permit to link different EPC models: process interfaces can be used to point from the end of a process to a subsequent process; hierarchical functions point from a function to a refining sub-process. Keller and Teufel (1998), Rump (1999), and van der Aalst (1999) provide formal approaches towards EPC syntax definition. Building

on this work, Nüttgens and Rump (2002) introduce 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 Schemas. Hierarchical EPC Schemas consist of flat EPC Schemas and hierarchy relations linking functions or process interface to other EPC Schemas. Figure 1 shows the example of a hierarchical EPC Schema for the waterfall model for software engineering as proposed in Boehm (1976). It consists of two processes: the *List requirements* function of the *Waterfall Model EPC* is linked via a hierarchical relation to the List requirements EPC process which describes a sub-process for this function.

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 and Scheer (1994), Langner et al. (1998), van der Aalst (1999), Rittgen (2000), and Dehnert (2002). In van der Aalst et al. (2002) the so called non-locality of join-connectors is presented as the major point of discussion. Recently, this aspect has been formalized by Kindler (2003). In this paper we will focus on EPC syntax based on the definition in Nüttgens and Rump (2002). Work on EPML started off in 2002 mainly inspired by heterogeneity of business process modelling tools and potential efficiency gains through the use of an intermediary format as described in e.g. Wüstner et al. (2002). As a first step, comparable efforts towards standardized interchange formats in the area of Petri Nets, BPML, and UML have been analyzed in Mendling and Nüttgens (2002). Work on syntactical correctness led to a revised EPC syntax definition based on implicit arc types and related syntax properties in Mendling and Nüttgens (2003a). In Mendling and Nüttgens (2003b) an analysis on the suitability of different XML schema languages for EPC syntax validation is presented. A proposal for an EPML schema is presented in Mendling and Nüttgens (2004). Mendling et al. (2004b) extend this version with elements to describe organizational entities, data elements, and applications. The EPML schema is available at http://wi.wu-wien.ac.at/~mendling/EPML.

3 Design Principles

In order to define a platform independent XML-based interchange format for EPC models, this global goal has to be translated into *general design principles*. These provide the foundation for design decisions. *XML design guidelines* help to develop XML Schemas in a standardized way and with certain quality properties.

3.1 EPML General Design Principles

In order to put EPML design principles into context, we present design principles proposed for the *ASC X12 Reference Model for XML Design (X12)* by

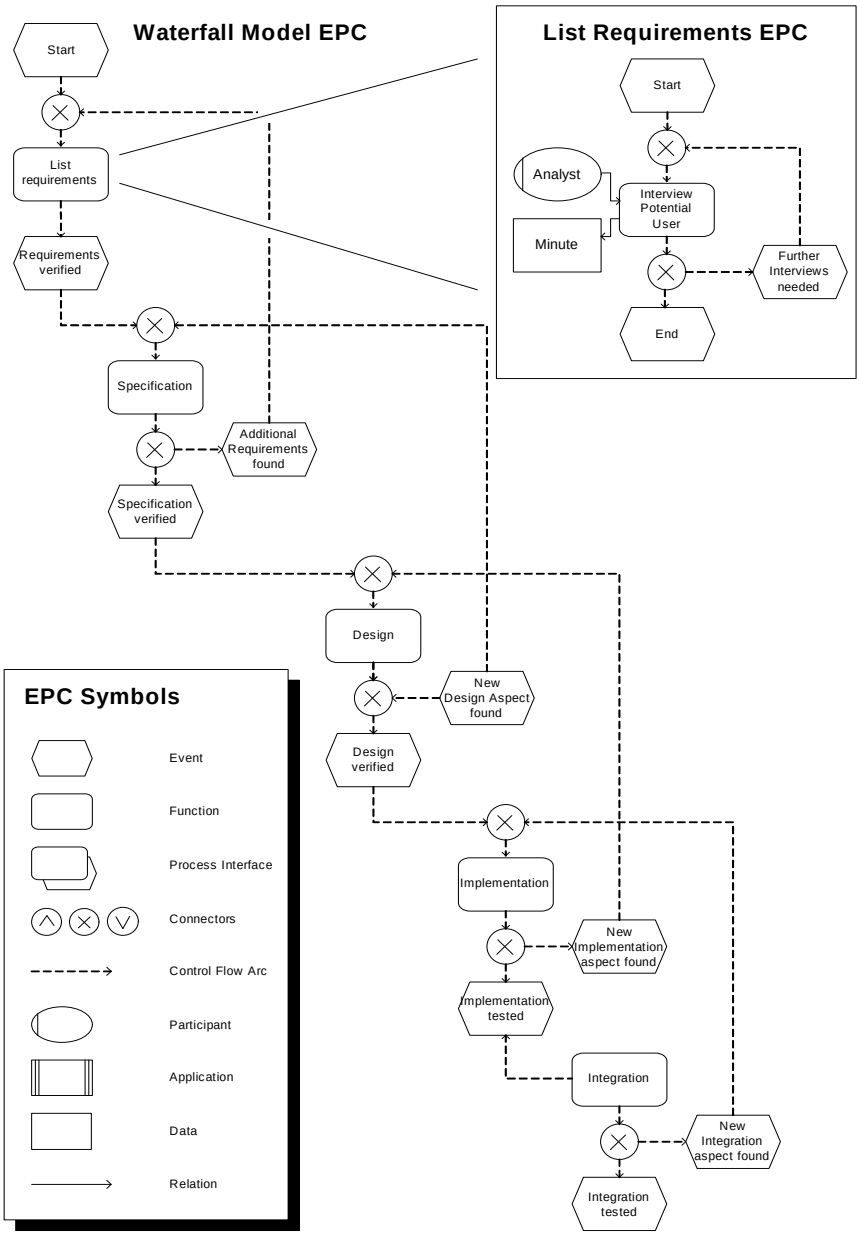


Figure 1 EPC representing the waterfall model for software engineering.

ANSI X12 (2002) and *Petri Net Markup Language (PNML)* by Billington et al. (2003). X12 is a specification describing a seven layer model for the development of business documents. The definition of X12 was guided by four high level design principles: alignment with other standards, simplicity, prescriptiveness, and limit randomness. *Alignment with other standards* refers to the specific domain of business documents where other organizations including OASIS and UN/CEFACT, World Wide Web Consortium, and OASIS UBL also develop specifications. *Simplicity* is a domain independent principle. It demands features and choices to be reduced to a reasonable minimum. *Prescriptiveness* is again related to business documents. This principle recommends one to define rather more precise and specific business documents than too few which are very general. *Limit Randomness* addresses certain constructs in XML schema languages that provide multiple options and choices. These aspects shall be limited to a minimum. The PNML approach by Billington et al. (2003) for Petri Nets is governed by the principles flexibility, no ambiguity, and compatibility. *Flexibility* is an important aspect for Petri Nets, because all kinds of currently discussed and prospective classes of Petri Nets shall be stored. This is achieved via labels which can be attached to arcs and nodes. *No ambiguity* refers to the problem of standardized labels. For this purpose Petri Net Type Definitions are used to define legal labels of a particular net type. *Compatibility* deals with the problem of semantically equivalent labels used by different Petri net types. These overlapping labels shall be exchangeable.

The EPML approach by Mendling and Nüttgens (2003c) reflects these different design principles. It is governed by the principles of readability, extensibility, tool orientation, and syntactical correctness . *Readability* expects EPML elements and attributes to have intuitive and telling names. This is important because EPML documents will be used not only by applications, but also by humans who write e.g. XSLT-scripts that transform between EPML and other XML vocabularies. Readability is partially related to simplicity and limited randomness of the X12 approach. *Extensibility* reflects a problem that is analogous to different types of Petri nets. Modelling of different business perspectives and views is an important aspect of BPM. EPML can express arbitrary perspectives instead of supporting a predefined set. *Tool orientation* deals with graphical representation of EPCs. This is a crucial feature, because BPM tools provide a GUI for developing models. EPML is able to store various layout and position information of EPC elements. Finally, *syntactical correctness* summarizes aspects dealing with EPC syntax elements and syntax constraints. As shown in Mendling and Nüttgens (2003b) XML schema languages do not provide a solution for this issue in the general case.

3.2 XML Design Guidelines

Basically, two general approaches towards XML design guidelines can be distinguished: a theoretical one building on normal forms and information

content measures like entropy; and a pragmatic one giving advise on when to use which XML language concepts and how to name elements and attributes.

The *theoretical approach* builds on insights from database theory. For relational database models concepts like functional dependency (FD), multivalue dependency (MVD), and join dependency (JD) have been formally described. For an overview see e.g. Biskup (1998). In order to derive schemas with good properties, decomposition algorithms have been developed to achieve different levels of normal forms. These normal forms avoid redundancies and anomalies from operations on relational data. Analogously, Embley and Mok (2001) as well as Arenas and Libkin (2002) present a normal form for XML documents called (XNF). In Arenas and Libkin (2003) an information-theoretic approach is presented that bridges the conceptual gap between relational and XML representations. A theory is developed building on entropy measures that brings forth a concept-independent understanding of the interrelation of redundancies and normal forms. A schema is called well-designed when it cannot contain instance data with an element that has less than maximum information in terms of conditional entropy. From this it can be shown that a schema which has only FDs and neither MVDs nor JD is well-designed iff (if and only if) it is in Boyce-Codd-Normal Form. FD for XML schemas occur when paths from the root to nodes in the XML tree depend upon other paths. Analogously, Arenas and Libkin (2003) show that an XML schema subject to FDs is well-designed iff it is in XNF. A violation of XNF implies redundancies in the sense that a path may reach different nodes, but that these nodes all have the same value. Such violations can be cured by a normalization algorithm also presented in Arenas and Libkin (2003) that moves attributes and creates new elements until XNF is achieved. Consequently, this implies for XML interchange format design that there should be no XPath (Clark and DeRose (1999)) statement that always returns a set of nodes all containing the same value. Then the XNF condition is fulfilled and the schema is well-designed.

Pragmatic approaches deal with extensibility and design leeway in XML. ISO (2001), SWIFT (2001), and ANSI X12 (2002) establish design rules in order to minimize ambiguity and maximize communicability of XML schemas. Pragmatic XML design guidelines include conventions for names; for the choice of style between elements and attributes; for the use of special schema language features; and for namespace support. *Naming conventions* refer to the choice of element and attribute names. ISO, SWIFT, and X12 agree on using English words for names. Names may also consist of multiple words in so-called Upper Camel Case (no separating space, each new word beginning with a capital letter). According to SWIFT and ISO, abbreviations and acronyms shall be limited to a minimum. *Style conventions* govern the choice between elements and attributes. X12 recommends the usage of attributes for metadata and elements for application data. In this context, it is a good choice to understand identifying keys as metadata and put them into attributes. That allows a DTD conforming usage of the ID,

IDREF, and IDREFS data types and a respective `key` or `keyref` declaration in a W3C XML Schema (Beech et al. (2001) and Biron and Malhorta (2001)). Furthermore, ANSI X12 (2002) considers attributes to provide a better readability of content. Therefore, content that can never be extended may also be put into attributes. *Schema conventions* recommend one to use only a reduced set of the expressive power provided by an XML schema language. X12 advises one to avoid mixed content, substitution groups, and group redefinition from another schema. One should use only named content types and built-in simple types, to name but a few aspects. We refer to ANSI X12 (2002) for a broader discussion. *Namespace conventions* refer to the usage of namespaces in instance documents. X12 recommends one to use explicit namespace references only at the root level. Theoretical and pragmatic approaches offer complementary guidelines for the development of "good" XML schemas. The guidelines presented have contributed to the EPML proposal.

4 Structure of EPML

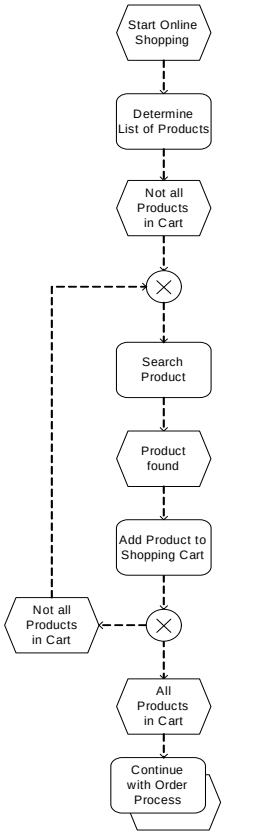
Figure 2 gives an overview of EPML and its syntax elements. The `epml` element is the root element of an EPML file. Like all other elements it may have `documentation` or `toolInfo` child elements. These elements may contain data that has been added by the editor of the EPML file or tool specific data attached by an application. They are defined in the schema as `anyType` which means that they may hold arbitrary data. It is recommended to add only such application specific data that has relevance for the internal storage of models in a certain tool, but which does no influence the graphical rendering of a model. General graphic settings may be defined in the `graphicsDefault` element. These apply for all EPC elements as long there is nothing else specified in an elements `graphics` sub-element. The `coordinates` element is meant to explicate the interpretation of coordinates annotated to graphical elements of an EPC. The `xOrigin` attribute may take the values `leftToRight` or `rightToLeft`, and the `yOrigin` attribute can hold `topToBottom` or `bottomToTop`. It is recommended to always use the `leftToRight` and `topToBottom` settings which most of the tools assume. Yet, there are still exceptions like Microsoft Visio that has its y-axis running from the bottom of the screen upward. It is recommended to transform these coordinates when storing EPC models in EPML.

In Nüttgens and Rump (2002) an EPC Schema Set is defined as a set of hierarchical EPC Schemas. Each of these hierarchical EPC Schemas consists of a flat EPC Schema which may have hierarchy relations attached with functions or process interfaces. In EPML a hierarchy of processes is organized by the help of directories. A `directory` element has a `name` attribute and it can contain other directories and/or EPC models. Each `epc` element is identified by an `epcId` attribute and has a `name` attribute. The `epcId` can be referenced by hierarchy relations attached to functions or process interfaces. In a hierarchy of EPC models there may be the problem of

```
epml
    documentation?
    toolInfo?
    graphicsDefault?
        fill? (color, image, gradient-color, gradient-rotation)
        line? (shape, width, color, style)
        font? (family, style, weight, size, decoration, color,
              verticalAlign, horizontalAlign, rotation)
    coordinates (xOrigin, yOrigin)
    definitions?
        definition* (defId)
            name
            description
    attributeTypes?
        attributeType* (typeId)
            description
    directory (name)
        directory* (name) ...
        epc* (epcId, name)
            attribute* (typeRef, value)
            event* (id, defRef) ...
            function* (id, defRef)
                name
                description?
                graphics?
                    position? (x, y, width, height)
                    fill? (color, image, gradient-color, gradient-rotation)
                    line? (shape, width, color, style)
                    font? (family, style, weight, size, decoration, color,
                           verticalAlign, horizontalAlign, rotation)
                syntaxInfo? (implicitType)
                toProcess? (linkToEpcId)
                attribute* (typeRef, value)
            processInterface* (id, defRef) ...
            and* ...
            or*  ...
            xor* ...
            arc* ...
            application* ...
            participant* ...
            dataField* ...
            relation* ...
```

Figure 2 EPML elements as a syntax tree. An asterisk denotes multiplicity of n, question marks refer to optional elements, attributes are put in brackets. The directory element may hold further directory and epc elements. The other epc sub-elements have similar child elements as the function element.

redundancy. An EPC process element might be used in two or more EPC models. In such a case there should be a place to store it once and reference it from the different models. This is achieved via the `definitions` element. It serves as a container for control flow elements that are used more than once in the model hierarchy. The `attributeTypes` element allows additional information to be defined. An `attributeType` element can specify such data that is not directly captured by standard elements of EPML. This may be important to model business views and perspectives on a process. Further EPML syntax elements that are also included in Figure 2 are introduced in the following section.




```
<?xml version ="1.0" encoding ="UTF-8"?>
<epml:epml xmlns:epml ="http://www.epml.de">

  <coordinates
    xOrigin ="leftToRight"
    yOrigin ="topToBottom "/>

  <directory name ="Root">

    <epc epcId ="1"
        name ="Online Shopping ">

      <event id ="1">
        <name>Start Online Shopping</name>
      </event>

      <arc id ="10">
        <flow source ="1" target ="2"/>
      </arc>

      <function id ="2">
        <name>Determine List of Products</name>
      </function >

      <arc id ="11">
        <flow source ="2" target ="3"/>
      </arc>

      <event id ="3">
        <name>Not all Products in Cart</name>
      </event>

      <arc id ="12">
        <flow source ="3" target ="4"/>
      </arc>

      <xor id ="4"/>

      <arc id ="13">
        <flow source ="4" target ="5"/>
      </arc>

      ...

      <processInterface id="111">
        <name>Continue with Order Process</name>

        ...

      </processInterface >

    </epc>

  </directory>

</epml:epml>
```

Figure 3 Flat EPC in graphical and EPML representation.

5 EPML in Detail

5.1 Flat EPCs in EPML

This section describes how a simple flat EPC process is encoded in EPML. Figure 3 shows the example of an *Online Shopping* process. The process starts with the event *Start Online Shopping* which is represented by an EPC `event` element. Afterwards the customer has to decide on a list of products that she wants to buy. This is modelled via a `function` element. It is a syntactical constraint of EPCs that functions and events have to alternate. The subsequent event triggers a loop that is modelled via a cycle between two `xor` elements. The loop continues until all products of the list have been found and added to the shopping cart. Finally, all products have been added to the shopping cart. A `processInterface` element *Continue with Order Process* points to a continuing process.

Figure 3 also illustrates the EPML representation of this EPC process. The root tag of every EPML file is `epml` and it must belong to the EPML namespace. In the example the directory tag contains one EPC model which has the name *Online Shopping* and the `id` attribute set to *1*. The EPC element serves as a container of an unordered set of EPC control flow elements. All of the latter have a unique `id` attribute. The name element of events and functions carry the text which is displayed as the label of the respective symbol in the process diagram. Arcs are modelled as individual elements with source and target attributes. This way of process graph representation is called edge element list according to Mendling and Nüttgens (2004). It is also used by Petri Net Markup Language (PNML) as described in Weber and Kindler (2002) or XML Metadata Interchange for UML models defined by OMG (2003) to name but a few. In contrast languages like the Business Process Execution Language for Web Services (BPEL4WS) specified by Andrews et al. (2003) use a block-oriented representation. AML, the XML format of ARIS Toolset defined by IDS Scheer AG (2001) uses adjacency sub-element lists that are attached to the source node of an arc. Arcs and connectors are not required to have a name. The connectors `and` and `or` are not included in the example of Figure 3. Syntactically, they are used similarly to the `xor` connector.

5.2 Hierarchical EPCs in EPML

Consider an extension of the example above. After the *Online Shopping* process has been modelled, the `function` *Search Product* is refined by a *Search Product* sub-process. This means that the EPML file now includes a second process and a hierarchical relation between the function and the sub-process. Figure 4 illustrates the EPML representation of EPC processes with hierarchy relations. The hierarchy relation is described via sub-element of the function called `toProcess`. That element has a `linkToEpcId` attribute pointing to the *Product Selection* process which has an `epcId` attribute of *2*. Hierarchy relations of `processInterfaces` are also described via a `toProcess` element. In that situation the process interface at the end of a process points to a start process interface of another process. The `epcId` attribute of a process must be unique for the whole EPML file. EPC models may be organized in a hierarchy of `directory` elements. Hierarchy relations between processes are allowed independently from where they are placed in the directory hierarchy, as long as hierarchy relations are acyclic. In order to avoid redundancies, multiple occurrences of a function, an event, or a process interface can be defined in the `definitions` block. In the example the event *Not all Products in Cart* occurs in multiple places in the process model. Semantically these events are the same. Therefore, an `definition` element can be defined to include data that is common for occurrences of that event, e.g. `name`. This avoids redundancies in the EPML file.

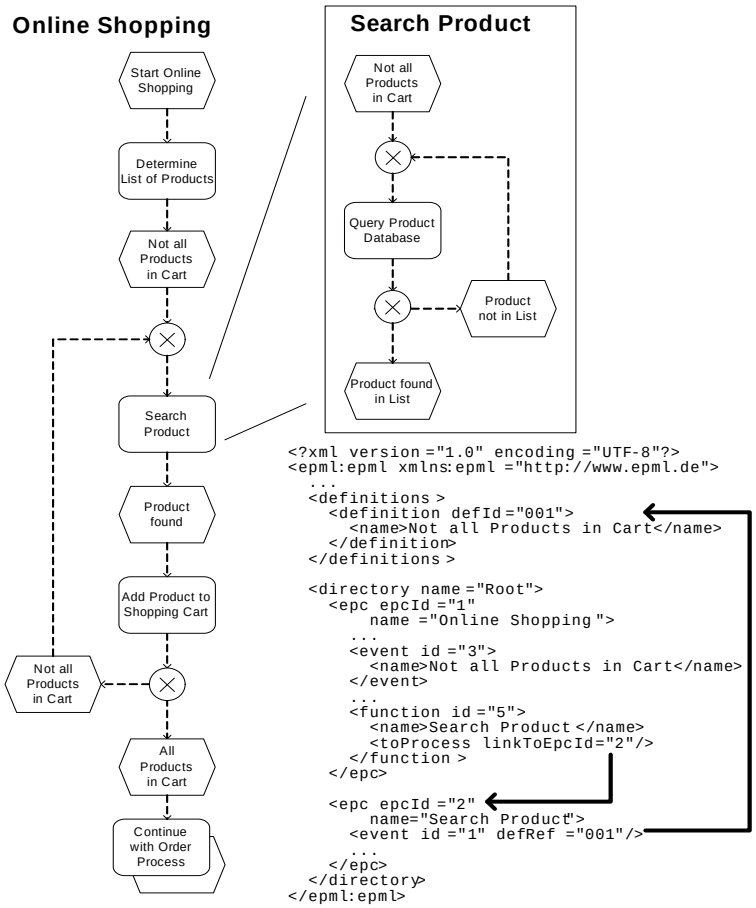


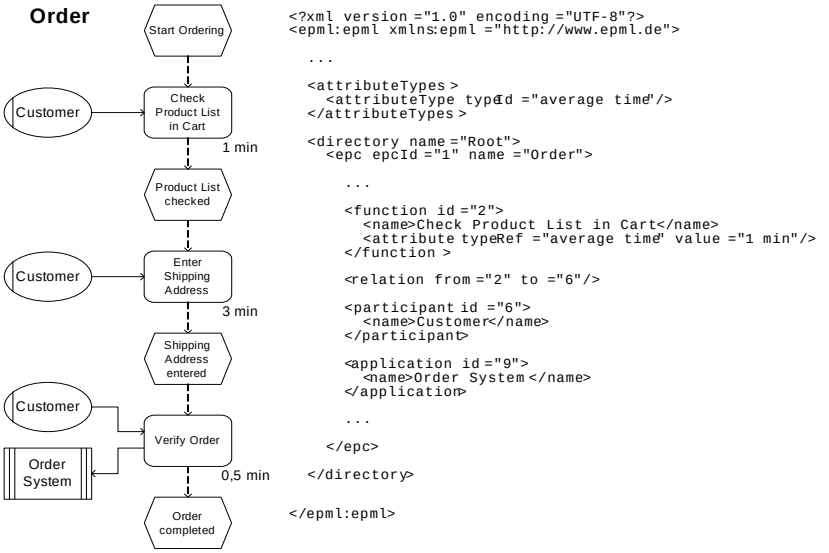
Figure 4 A hierarchical EPC schema with two linked EPC Processes and an event definition.

5.3 Business Perspectives and Views in EPML

Business perspectives and views play an important role for the analysis and conception of process models, especially for EPCs. Perspectives have proven valuable to partition the specification of a complex system (Finkelstein et al. (1992)). There have been many different perspectives proposed for business process modelling. In Scheer (2000) the Architecture of Integrated Information Systems (ARIS) extends the EPC with a data-oriented, a functional, an organizational, an application-oriented, and a product/service-oriented perspective. The PROMET concept by Österle (1995) differentiates between business dimensions explicitly including organization, data, functions, and personnel. Rosemann and zur Mühlen (1997) give an in-depth survey of organizational entities provided in workflow management systems. In Neumann and Strembeck (2002) the link between role-based access control

(RBAC) and business scenarios is analyzed in order to define a methodology to generate role hierarchies. From a delegation perspective van der Aalst et al. (2003) formalize the organizational perspective of a workflow system via a meta model including resources, organizational units, users, and roles. In White (2003)swim lanes and pools are recommended as a metaphor for the graphical representation of parties involved in a process. Recently, BPM languages like BPEL4WS contain references to WSDL descriptions of Web Services as a new category of resource perspectives. The OWL Services Coalition (2004) has developed a standardized business process ontology for Web Service. Thinking of the variety of potential perspectives and views the definition of such an ontology is a difficult. There are even doubts whether a standardized ontology is desirable, because different domains and different business sectors need tailor-made meta models that best fit their specific business model (see Karagiannis and Kühn (2002)).

In general there are two categories of information that are frequently added to a business process model. Firstly, *attributes* represented as *(name, value)* pairs can be used to attach statistical or configurational data to a process or to process objects. In EPML they reflect the design principle of extensibility, as arbitrary `attributeTypes` can be declared at the top of an EPML file. Single `attribute` elements can be attached to `epc` elements and to all its child elements like e.g. `function` elements. The `attributeType` may have a `description` and it must have a unique `typeId` attribute. This type name is referenced in the `typeRef` attribute of an `attribute` element. The attribute type declaration provides for a consistent naming of extension attributes used by individual tools. Secondly, various *objects* involved in the execution of a business process are frequently displayed as icons in the visual process model. For this purpose, EPML includes non-control flow elements which can be displayed in a graphical EPC process model. In contrast to some business process modelling tools that offer various icons and object type, EPML restricts itself to three objects: `dataField`, `participant`, and `application`. These three process objects are also found in both XPDL of the WfMC (2002) and, with different names, the Architecture of Integrated Information Systems (ARIS) proposed by Scheer (2000). All these three elements have a `name` and a `description` element and they are identified by a unique `id` attribute. Furthermore, they may have `graphics` and `attribute` elements. Relationships between these elements or between these elements and control flow elements are represented by `relation` elements. A `relation` is a directed edge between an element whose `id` is referenced in the `from` attribute and another element referred to in the `to` attribute. The `relation` element is related to the `arc` element. Yet, the syntactical distinction between both allows to easily identify control flow with `arc` elements. The `relation` elements may have multiple `graphics` elements. They can also contain `attribute` elements. Furthermore, the `defRef` attribute of a relation must reference a `defId` of a `definition` element. The definition is meant to describe the semantics of the relationship. Including these new elements an `epc` may have `function`, `event`, `processInterface`, `and`, `or`, `xor`, and




```
<?xml version="1.0" encoding="UTF-8"?>
<epml:epml xmlns:epml="http://www.epml.de">

  ...

  <attributeTypes>
    <attributeType typeId="average time"/>
  </attributeTypes>

  <directory name="Root">
    <epc epcId="1" name="Order">

      ...

      <function id="2">
        <name>Check Product List in Cart</name>
        <attribute typeRef="average time" value="1 min"/>
      </function>

      <relation from="2" to="6"/>

      <participant id="6">
        <name>Customer</name>
      </participant>

      <application id="9">
        <name>Order System</name>
      </application>

      ...

    </epc>

  </directory>

</epml:epml>
```

Figure 5 An EPC with non control flow aspects.

`arc` elements as well as `participant`, `application`, `dataField`, and `relation` elements as children.

Figure 5 illustrates the use of `attribute` and non control flow elements in an *Order* process. In the header of the EPML file an attribute type with a *typeId* of *average time* is declared. An attribute with value *1 min* is annotated to the function *Check Product List in Cart*. Furthermore, a participant called customer is modelled as a contributor to each of the functions, as well as an application *Order System* that receives data from the *Verify Order* function. Both, attribute types and non control flow elements can be used to declare arbitrary business perspectives for an EPC business process in EPML.

5.4 Graphical Information in EPML

Graphical Information refers to the rendering of EPC models in graphical BPM tools. This topic is not specific to EPML. The Petri Net Markup Language (PNML) has worked out and included a proposal for graphical information to be exchanged between modelling tools. This concept is also well suited for EPML and adopted here. There are some small modifications that will be made explicit in the discussion of the details. Each control flow object may have a `graphics` element. Similar to the top level element `graphicsDefault` it may contain `fill`, `line`, and `font` information. Additionally, `position` information may be included for each control flow element.

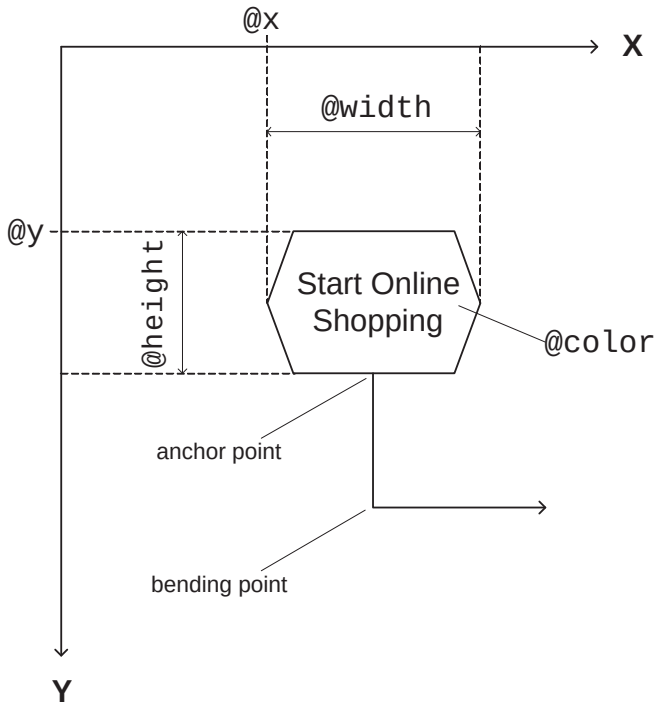


Figure 6 An Event and an Arc with Graphical Information.

All the four attributes of the `position` element refer to the smallest rectangle parallel to the axes that can be drawn to contain the whole polygon symbolizing the object (see Figure 6). The `x` and `y` attributes of the object describe the offset from the origin of the coordinates system of that angle of the object that is closest to the origin. The `width` and the `height` describe the length of the edges of the container rectangle. In PNML a separate dimension element is used to represent width and height. Arcs may have multiple position elements to describe anchor point where the arc runs through. Position elements of arcs do not have width and height attributes. An arc should have at least two `position` elements for two *anchor points*. Each `position` in between describes a *bending point*.

The `fill` element describes the rendering of the interior of an object. Arcs do not have fill elements. The `color` attribute must take a RGB value or a predefined color of Cascading Stylesheets 2 (CSS2) (Bos et al. (1998)). In order to describe a continuous variation of the filling color an optional `gradient-color` may be defined. The `gradient-rotation` sets the orientation of the gradient to vertical, horizontal, or diagonal. If a URI of an image is assigned to `image` the other attributes of `fill` are ignored. The `line` element defines the outline of an object. The `shape` attribute refers to how arcs are displayed: the value *line* represents a linear connection of anchor points to

form a polygon; the value *curve* describes a quadratic Bezier curve. The `font` element holds `family`, `style`, `weight`, `size`, and `decoration` attributes in conformance with CSS2. In addition to PNML, there may be a font color defined. The `verticalAlign` and `horizontalAlign` attributes specify the alignment of the text. The align attribute in PNML corresponds to the EPML `horizontalAlign` attribute, and `verticalAlign` is covered by a PNML offset element. The `rotation` attribute describes a clockwise rotation of the text similar to the concept in PNML.

5.5 Syntactical Correctness of EPCs in EPML

Syntactical correctness of EPCs depends on a set of properties described in Nüttgens and Rump (2002). These include process graph related properties; cardinality constraints on the number of incoming and outgoing arcs for each control flow element; type consistency constraints prescribing alternation of functions and events with an arbitrary number of intermediate connectors; and properties of hierarchy relations. Mendling and Nüttgens (2003b) show that most of these syntax rules can be validate by the help of XML schema languages. However, some properties require graph expansion and transitive closure calculation like for example the prohibition of cycles that consist only of connector elements. Such so called graph expansion properties cannot be validated by usual XML schema languages.

Type consistency also belongs to the properties that require closure calculation because alternating functions and events may have multiple intermediate connectors. But in contrast to other graph expansion properties type consistency can be redefined in a way so it can be validated by some XML schema languages. In Mendling and Nüttgens (2003a) for each EPC control flow element a set of so called implicit types is defined. An implicit type indicates in which constellation a control flow element is used in a process model. For example an AND connector can be either a join or a split operation and it may either follow event(s) or function(s). Each of the four resulting constellation defines one implicit type of the AND connector. Similarly, arcs have two different implicit types. So called function-event-arcs (FEA) run from functions to events. Event-function-arcs (EFA) are defined analogously. For a complete discussion see Mendling and Nüttgens (2003a).

Figure 7 illustrates the type consistency problem. Connector *AND3* is only connected with other connectors (*AND1*, *AND2*, and *AND4*). In order to check whether *AND3* has antecedants that are all of event type and descendants that are all of function type, or the other way round, the graph needs to be expanded. As all (transitive) antecedants of *AND3* are events and all (transitive) descendants are of function type, the connector complies with EPC type consistency rules. In Figure 7 (b) all control flow elements are labelled with their implicit type. As all incoming arcs of *AND3* are of implicit type EFA and all outgoing arcs as well, the connector fulfills the type consistency rules described in Mendling and Nüttgens (2003a)

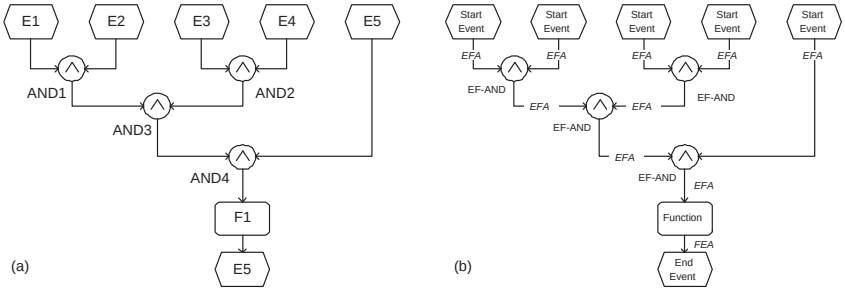


Figure 7 An EPC process model (a) without implicit types and (b) with implicit types for each control flow element.

without having to expand the process graph. In EPML each control flow element may have an optional `<syntaxInfo>` element with an `@implicitType` attribute. This attribute can be used by validation tools to speed up syntax checking. Furthermore, it can be used to attach information generated during validation of type consistency rules.

6 Summary

This paper has presented an XML-based interchange format for Event-Driven Process Chains called EPML. EPML builds on EPC syntax related work and reflects experiences from the specification of other XML vocabularies in its design principles. EPML supports:

- the definition of flat EPC models arranged in directories;
- the specification of hierarchical EPCs including element definitions;
- the attachment of business perspectives and views to functions;
- the exchange of detailed graphical information; and
- the specification of syntax-related information.

EPML addresses the lack of interchange formats in the domain of business process modelling. XSLT scripts can be programmed to provide for a simple conversion between EPML and other XML-based formats. Examples of such transformations between EPML and AML of ARIS Toolset as well as EPML and SVG are reported in Mendling et al. (2004a) and Mendling et al. (2004b), respectively. Future areas of research will be dedicated to the interrelation of different business process modelling methods. The *XML4BPM Workshop* (Nüttgens and Mendling (2004)) that gathered experts working on XML-based interchange formats has been a first step in this direction. One open question in this context is: how can separately developed interchange formats be integrated in a meaningful way. Another research aspect is related to tool support for EPML. The success of EPML will depend on the number and analysis capabilities of the tools supporting it. So far, Cuntz and Kindler (2004) have developed an EPC simulation tool that uses

EPML as an interchange format. Furthermore, EPML is supported by the professional BPM tool Semtalk. Finally, there is still much research needed to create a general understanding of business perspectives for BPM. In this context, the development of EPML can - beyond its principle purpose as an interchange format - serve as a catalyst and a framework for the discussion of all these related topics.

References

van der Aalst WMP (1999) Formalization and Verification of Event-driven Process Chains. *Information and Software Technology*, 41(10):639–650.

van der Aalst WMP, Desel J, Kindler E (2002) On the semantics of EPCs: A vicious circle. In Nüttgens M, Rump FJ, eds., *Proc. of the 1st GI-Workshop on Business Process Management with Event-Driven Process Chains (EPK 2002), Trier, Germany*, pages 71–79.

van der Aalst WMP, Kumar A, Verbeek HMW (2003) Organizational Modeling in UML and XML in the Context of Workflow Systems. In *Proceedings of the 2003 ACM Symposium on Applied Computing (SAC)*, pages 603–608.

Andrews T, Curbera F, Dholakia H, Goland Y, Klein J, Leymann F, Liu K, Roller D, Smith D, Thatte S, Trickovic I, Weerawarana, S (2003) *Business Process Execution Language for Web Services, Version 1.1*, Specification, BEA Systems, IBM Corp., Microsoft Corp., SAP AG, Siebel Systems.

ANSI X12 (2002) *ASC X12 Reference Model for XML Design*, Technical Report Type II - ASC X12C/TG3/2002-xxx, ANSI ASC X12C Communications and Controls Subcommittee.

Arenas M, Libkin L (2002) A normal form for XML documents. In *Proceedings of the 21st ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'02)*, pages 85–96.

Arenas M, Libkin L (2003) An Information-Theoretic Approach to Normal Forms for Relational and XML Data. In *Proceedings of the 22nd ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'03)*, pages 15–26.

Beech D, Lawrence S, Moloney M, Mendelsohn N, Thompson, HS (2001) *XML Schema Part 1: Structures*. W3C Recommendation 02 May, World Wide Web Consortium.

Billington J, Christensen S, van Hee KE, Kindler E, Kummer O, Petrucci L, Post R, Stehno C, Weber M (2003). The Petri Net Markup Language: Concepts, Technology, and Tools. In van der Aalst WMP, Best E, eds., *Applications and Theory of Petri Nets 2003, 24th International Conference, ICATPN 2003, Eindhoven, The Netherlands*, Vol. 2679 of *Lecture Notes in Computer Science*, pages 483–505.

Biskup J (1998) Achievements of relational database schema design theory revisited. In Libkin L, Thalheim B, eds., *Semantics in Databases*, Vol. 1358 of *Lecture Notes in Computer Science*, pages 29–54.

Biron PV, Malhorta, A (2001) *XML Schema Part 2: Datatypes.* W3C Recommendation 02 May, World Wide Web Consortium.

Boehm BW (1976) Software Engineering *IEEE Transactions on Computers*, 25(12):1226–1241.

Bos B, Lie HW, Lilley C, Jacobs I (1998) *Cascading Style Sheets, Level 2.* W3C Recommendation 12-May-1998, World Wide Web Consortium.

Chen R, Scheer AW (1994) *Modellierung von Prozessketten mittels Petri-Netz-Theorie*, Technical Report 107, Institut für Wirtschaftsinformatik, Saarbrücken, Germany.

Christensen E, Curbera F, Meredith G, Weerawarana S (2001). *Web Service Description Language (WSDL) 1.1*, W3C Note.

Clark J (1999) *XSL Transformations (XSLT) Version 1.0.* W3C Recommendation 16 November, World Wide Web Consortium.

Clark J, DeRose S (1999) *XML Path Language (XPath) Version 1.0.* W3C Recommendation 16 November, World Wide Web Consortium.

Cuntz N, Kindler E (2004) On the semantics of EPCs: Efficient calculation and simulation. In Nüttgens M, Rump FJ, ed., *Proc. of the 3rd GI-Workshop on Business Process Management with Event-Driven Process Chains (EPK 2004), Luxembourg, Luxembourg*, pages 7–26.

Dehnert J (2002) Making EPCs fit for Workflow Management. In Nüttgens M, Rump FJ, eds., *Proc. of the 1st GI-Workshop on Business Process Management with Event-Driven Process Chains (EPK 2002), Trier, Germany*, pages 51–69.

Delphi Group (2003) *BPM 2003 – Market Milestone Report, Delphi Group White Paper.*

Embley DW, Mok WY (2001) Developing XML documents with guaranteed "good" properties. In Kunii HS, Jajodia S, Sølvberg, A, eds., *Conceptual Modeling - ER 2001, Proc. of the 20th International Conference on Conceptual Modeling*, Vol. 2224 of *Lecture Notes in Computer Science*, pages 426–441.

Ferraiolo J, Jun F, Jackson D (2003) *Scalable Vector Graphics (SVG) 1.1.* W3C Recommendation 14 January 2003, World Wide Web Consortium.

Finkelstein A, Kramer J, Nuseibeh B, Finkelstein L, Goedicke M (1992) Viewpoints: A Framework for Integrating Multiple Perspectives in System Development. *International Journal of Software Engineering and Knowledge Engineering*, 2(1):31–57.

Gartner Research (2002) *The BPA Market Catches Another Major Updraft*, Gartner's Application Development & Maintenance Research, 12 June, Note M-16-8153.

IDS Scheer AG (2001) *XML-Export und -Import mit ARIS 5.0, Whitepaper January 2001.*

ISO - International Organization for Standardization (2001). *XML Design Rules.* ISO 15022 Technical Specification.

Karagiannis D, Kühn H (2002) Metamodelling Platforms. Invited Paper. In Bauknecht K, Min Tjoa A, Quirchmayer, G, eds., *Proceedings of the 3rd International Conference EC-Web 2002 - Dexa 2002, Aix-en-Provence,*

France, Vol. 2455 of *Lecture Notes in Computer Science*, pages 182–196.

Keller G, Nüttgens M, Scheer AW (1992). *Semantische Prozessmodellierung auf der Grundlage "Ereignisgesteuerter Prozessketten (EPK)"*, Technical Report 89, Institut für Wirtschaftsinformatik, Saarbrücken, Germany.

Keller G, Meinhardt S (1994) *SAP R/3 Analyzer. Business process reengineering based on the R/3 reference model.* SAP AG.

Keller G, Teufel T (1998). *SAP(R) R/3 Process Oriented Implementation: Iterative Process Prototyping.* Addison-Wesley.

Kindler E (2003) On the semantics of EPCs: A framework for resolving the vicious circle (Extended Abstract). In Nüttgens M, Rump FJ, eds., *Proc. of the 2nd GI-Workshop on Business Process Management with Event-Driven Process Chains (EPK 2003), Bamberg, Germany*, pages 7–18.

Langner P, Schneider C, Wehler J (1998) Petri Net Based Certification of Event driven Process Chains. In Desel J, Silva M, eds., *Application and Theory of Petri Nets*, Vol. 1420 of *Lecture Notes in Computer Science*, pages 286–305.

Mendling J, Brabenetz, A, Neumann G (2004) EPML2SVG - Generating Websites from EPML Processes. In Nüttgens M, Rump FJ, ed., *Proc. of the 3rd GI-Workshop on Business Process Management with Event-Driven Process Chains (EPK 2004), Luxembourg, Luxembourg*, pages 55–64.

Mendling J, Nüttgens M (2002) Event-Driven-Process-Chain-Markup-Language (EPML): Anforderungen zur Definition eines XML-Schemas für Ereignisgesteuerte Prozessketten (EPK). In Nüttgens M, Rump FJ, eds., *Proc. of the 1st GI-Workshop on Business Process Management with Event-Driven Process Chains (EPK 2002), Trier, Germany*, pages 87–93.

Mendling J, Nüttgens M (2003a) EPC Modelling based on Implicit Arc Types. In Godlevsky M, Liddle SW, Mayr HC, eds., *Proc. of the 2nd International Conference on Information Systems Technology and its Applications (ISTA), Kharkiv, Ukraine*, Vol. 30 of *Lecture Notes in Informatics*, pages 131–142.

Mendling J, Nüttgens M (2003b) EPC Syntax Validation with XML Schema Languages. In Nüttgens M, Rump FJ, eds., *Proc. of the 2nd GI-Workshop on Business Process Management with Event-Driven Process Chains (EPK 2003), Bamberg, Germany*, pages 19–30.

Mendling J, Nüttgens M (2003c) XML-basierte Geschäftsprozessmodellierung. In Uhr W, Esswein W, Schoop E, eds., *Proc. of Wirtschaftsinformatik 2003 / Band II, Dresden, Germany*, pages 161 –180.

Mendling J, Nüttgens M (2004) XML-based Reference Modelling: Foundations of an EPC Markup Language. In Becker J, ed., *Referenzmodellierung - Proceedings of the 8th GI-Workshop on Reference Modelling, MKWI Essen, Germany*, pages 51–71.

Mendling J, Neumann G, Nüttgens M (2004) Transformation of ARIS Markup Language to EPML. In Nüttgens M, Rump FJ, ed., *Proc. of the 3rd GI-Workshop on Business Process Management with Event-Driven Process Chains (EPK 2004), Luxembourg, Luxembourg*, pages 27–38.

Neumann G, Strembeck M (2002) A Scenario-driveen Role Engineering Process for Functional RBAC Roles. In *Proc. of the 7th ACM Symposium on Access Control Models and Technologies (SACMAT), Monterey, USA, June 2002.*

Nüttgens M, Mendling J (2003) *XML4BPM 2004 - Proceedings of the 1st GI-Workshop of XML Interchange Formats for Business Process Management, Marburg, Germany.*

Nüttgens M, Rump FJ (2002) Syntax und Semantik Ereignisgesteuerter Prozessketten (EPK). In Desel J, Weske M, eds., *Promise 2002 - Proceedings of the GI-Workshop, Potsdam, Germany*, Vol. 21 of *Lecture Notes in Informatics*, pages 64–77.

OMG Object Management Group (2003) *XML Metadata Interchange (XMI)*, Specification, Version 2.0.

Österle H (1995) *Business Engineering.* Springer Verlag.

OWL Services Coalition (2004). *OWL-S: Semantic Markup for Web Services.* Whitepaper Version 1.0.

Rittgen P (2000) Paving the Road to Business Process Automation. In *Proc. of the European Conference on Information Systems (ECIS), Vienna, Austria*, pages 313–319.

Rump FJ (1999) *Geschäftsprozessmanagement auf der Basis ereignisgesteuerter Prozessketten - Formalisierung, Analyse und Ausführung von EPKs.* Teubner Verlag.

Rosemann M, zur Mühlen M (1997) Evaluation of Workflow Management Systems - a Meta Model Approach. In Siau K, Wand Y, Parsons J, eds., *Proc. of the 2nd CAiSE/IFIP 8.1 International Workshop on Evaluation of Modeling Methods in Systems Analysis and Design (EMMSAD'97), Barcelona, Spain, June 1997.*

Scheer AW (2000) *ARIS Business Process Modelling.* Springer Verlag.

SWIFT Society for Worldwide Interbank Financial Telecommunication (2001) *SWIFT Standards XML Design Rules Version 2.3*, Technical Specification.

Weber M, Kindler E (2002) The Petri Net Markup Language. In Ehrig H, Reisig W, Rozenberg G, Weber H, eds., *Petri Net Technology for Communication Based Systems*, Vol. 2472 of *Lecture Notes in Computer Science*, pages 124–144.

WfMC Workflow Management Coalition (2002). *Workflow Process Definition Interface – XML Process Definition Language*, Document Number WFMC-TC-1025, October 25, 2002, Version 1.0.

White SA (2003) Business Process Modeling Notation. Working Draft 1.0, Business Process Management Initiative.

Wüstner E, Hotzel T, Buxmann P (2002). Converting Business Documents: A Classification of Problems and Solutions using XML/XSLT. In *Proceedings of the 4th International Workshop on Advanced Issues of E-Commerce and Web-based Systems (WECWIS)*, pages 61–68.