Markus Nüttgens, Jan Mendling (eds.)

XML4BPM 2004

XML Interchange Formats for Business Process Management

1st Workshop of German Informatics Society e.V. (GI)
in conjunction with the 7th GI Conference "Modellierung 2004"

March 25, 2004 in Marburg (Germany)

Proceedings

Organizer

This workshop is organized by the GI Working Group "Business Process Management with Event-Driven Process Chains (EPC)" within the GI Special Interest Group WI-MobIS (FB-WI) in conjunction with the 7th GI Conference "Modellierung 2004".

Dr. Markus Nüttgens
Email: markus@nuettgens.de

Dipl.-Wirt.-Inf. Dipl.-Kfm. Jan Mendling
Email: jan.mendling@wu-wien.ac.at

XML4BPM 2004 / XML Interchange Formats for Business Process Management. Eds.: Markus Nüttgens, Jan Mendling – Marburg 2004.

Preface

This proceedings volume contains the papers presented at the 1st GI workshop XML Interchange Formats for Business Process Management (XML4BPM 2004) which is held in conjunction with the 7th GI Conference "Modellierung 2004" taking place in Marburg (Germany) from March 24 to 26, 2004. The workshop aims to discuss recent topics concerning XML based domain and interchange formats for Business Process Management on a broad conceptual and technical basis and in dialogue between research and industry.

Beyond the presentation of the state-of-the-art the workshop covers the concepts behind interchange formats designed in research and industry and identifies perspectives for integration and future research. The topics in detail include the following:

- Interchange Formats for Event-Driven Process Chains (EPC),
- Interchange Formats for Petri Nets,
- Interchange Formats for UML,
- Interchange Formats for Graphs,
- Interchange Formats for Web Service Compositions,
- Interchange Formats for Business Process Collaborations.

This proceedings volume includes six carefully selected papers presented at the workshop that illustrate the concepts behind BPEL4WS, XMI, PNML, EPML, ebXML BPSS and GXL. Papers could be submitted by invitation only in order to acquire high quality and up-to-date contributions from research and industry and to provide a forum for interested persons especially from German speaking countries.

The papers focus on the concepts and touch aspects including methodical and conceptual background, design principles, meta-models, application scenarios, tool support, available functionality as well as experiences, problems, open issues and future directions.

We thank the authors, the members of the program committee, and the local organization team of the GI Conference "Modellierung 2004" for their contributions to the realization of this workshop.

Saarbrücken and Wien, March 2004

Markus Nüttgens
Jan Mendling

Program Committee

Christian Huemer, University of Vienna, Austria
Mario Jeckle, University of Applied Sciences Furtwangen, Germany
Ekkart Kindler, University of Paderborn, Germany
Frank Leymann, IBM Software Group, Böblingen, Germany
Jan Mendling, Vienna University of Economics and BA, Austria
Markus Nüttgens, University of Saarland, Germany (Chair)
Andreas Winter, University of Koblenz, Germany

Organization

Jan Mendling, Vienna University of Economics and BA, Austria
Markus Nüttgens, University of Saarland, Germany

Table of Contents

Modeling Business Processes with BPEL4WS

Frank Leymann, Dieter Roller

IBM Software Group
Schönaicher Strasse 220
71032 Böblingen
ley1@de.ibm.com
rol@de.ibm.com

Abstract: Business Process Execution Language for Web Services (BPEL4WS) allows defining both, business processes that make use of Web services, and business processes that externalize their functionality as Web services. This short paper introduces the basic language elements of BPEL4WS using a simple example. The concepts underlying the language are briefly explained: Establishing bilateral partnerships, correlating messages and processes, defining the order of the activities of a business process, event handling, handling exceptions via long-running transactions, the resulting programming model, and the usage of BPEL4WS in pure B2B scenarios.

1 Introduction

Web services are components, which are based on the industry standards WSDL [1], UDDI [2], and SOAP [3]. They enable to connect different components even across organizational boundaries in a platform and language independent manner [4].

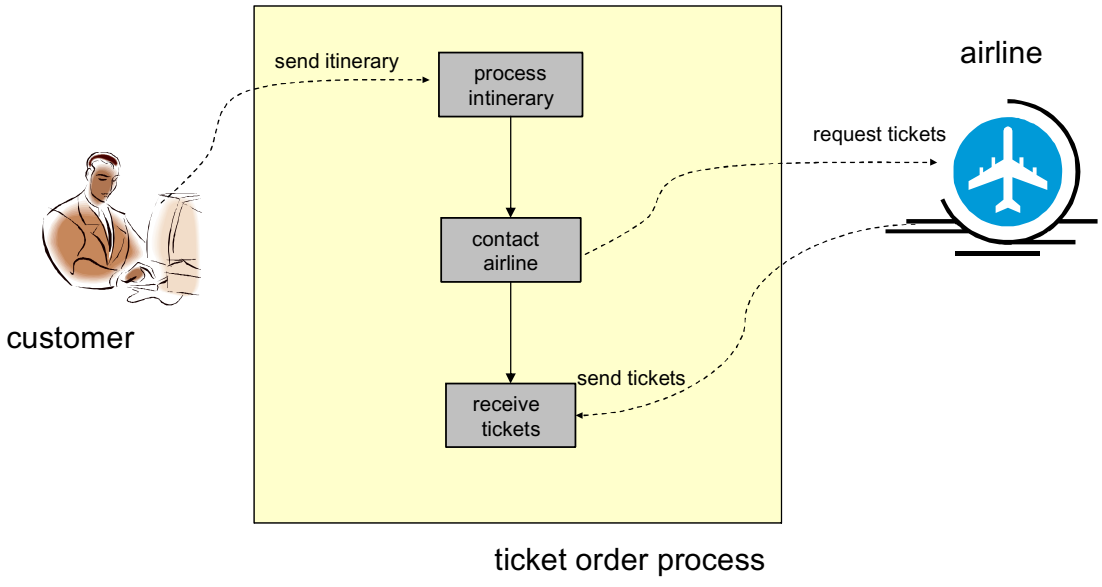
None of these standards for Web services however provides for the definition of the business semantics of Web services, the Web services are isolated and opaque. Braking isolation means to connect Web services and specify how collections of Web services are jointly used to realize more complex functionality – typically a business process. A "business process" specifies the potential execution order of operations from a collection of Web services, the data shared between these Web services, which partners are involved and how they are involved in the business process, joint exception handling for collections of Web services etc. In particular, the capability of support for long-running transactions between Web services increases consistency and reliability for Web services applications. Breaking opaqueness of Web services means specifying usage constraints of operations of a collection of Web services and their joint behavior – this is obviously very similar to specifying business processes.

Business Process Execution Language for Web Services [5] (BPEL4WS or BPEL for short) allows specifying business processes and how they relate to Web services. This includes specifying how a business process makes use of Web services to achieve its goal, and it includes specifying Web services that are provided by a business process. Business processes specified in BPEL are fully executable and they are portable between BPEL conformant environments. A BPEL business process interoperates with the Web services of its partners, whether these Web services are realized based on BPEL or not. Finally, BPEL supports the specification of business protocols between partners and views on complex internal business processes.

BPEL combines WSFL [6] and XLANG [7], superseding the corresponding specifications. The first version BPEL4WS V. 1.0. has been published in August 2002, a second version BPEL4WS V. 1.1. in May 2003 as input for the standardization within OASIS. The appropriate technical committee [8] is working since the time of submission and has given itself the charter to complete a first version of the standard by middle of 2004.

2 A First Look

The simple business process sketched in the figure helps illustrating the basic elements of BPEL. A travel agent specifies a business process that supports the travel agent in managing airline ticket requests that are requested from customer by sending in an itinerary. After the itinerary has been received, the airline is contacted with an appropriate ticket request. Then the process waits until the airline sends the tickets.



For simplicity of the business process, it is assumed that the tickets will be picked up by the customer. BPEL does not specify the graphical representation of business processes; appropriate methods for visually representing business processes are currently being developed; one of them is Business Process Modeling Notation (BPMN) [9].

This simple process is defined via BPEL as shown in lines 1 to 46. The travel agent gives the business process the name `ticketOrder` (line 1). The different tasks include the receiving of the itinerary (lines 23 to 29), passing the customer's itinerary to an airline requesting corresponding tickets (lines 30 to 37), and finally receiving the requested tickets from the airline (lines 38 to 44). For simplicity of the business process, it is assumed that the tickets will be picked up by the customer in person.

The set of relationships with partners that the agent's process maintains are defined in lines 2 to 11: Lines 3 to 6 introduce the relationship with the partner "customer", and lines 7 to 10 introduce the relationship with the partner "airline". A partner link identifies a relationship between a process and a partner and specifies the Web services mutually used by the partner or process, respectively (see section 3 for more details).

The messages that are persisted by the process are called "variables" (line 12 to 17). Variables are WSDL messages that are typically received from or sent to partners (see section 5 for more details). For example, the process stores an `itineraryMessage` as an `itinerary` variable. The `itineraryMessage` is received from the `customer` (line 23) when the customer uses the `sendItinerary` operation of the processes itinerary port type (lines 25 and 26). This message is stored into the `itinerary` variable (line 27) once received. When the process passes on the `itinerary` message to the `airline` (line 30) by using the `requestTicket` operation of the `ticketOrder` port type (lines 32 and 33) offered by the airline, this message is a copy of the `itinerary` variable (line 34).

The usage of an operation in a business process is called an "activity" (see section 5 for more details). To define the order in which the activities have to be performed, the `ticketOrder` process structures its activities as a `flow` (line 18): A flow is a directed graph with the activities as nodes and so-called links as edges connecting the activities. The `links` required to define the flow between different `ticketOrder` process' activities are specified in lines 19 to 22. The activities then specify whether they are the source or the target of one or more links defined via a link. For example, the `receive` activity of line 23 is the `source` of the `order-to-airline` link (line 20) with the `invoke` activity of line 30 being the `target` (line 35).

```
1    <process name="ticketOrder">

2       <partnerLinks>
3          <partnerLink name="customer"
4                       partnerLinkType="agentLink"
5                       myRole="agentService"
6                       partnerRole="customer"/>
7          <partnerLink name="airline"
8                       partnerLinkType="buyerLink"
9                       myRole="ticketRequester"
```

```
10                      partnerRole="ticketService"/>
11     </partnerLinks>

12     <variables>
13        <variable name="itinerary"
14                  messageType="itineraryMessage"/>
15        <variable name="tickets"
16                  messageType="ticketsMessage"/>
17     </variables>

18     <flow>

19        <links>
20           <link name="order-to-airline"/>
21           <link name="airline-to-agent"/>
22        </links>

23        <receive name=”processItinerary”,
24                 partnerLink="customer"
25                 portType="itineraryPT"
26                 operation="sendItinerary"
27                 variable="itinerary"
28           <source linkName"order-to-airline"/>
29        </receive>

30        <invoke  name=”contactAirline”,
31                 partnerLink="airline"
32                 portType="ticketOrderPT"
33                 operation="requestTickets"
34                 variable="itinerary">
35            <target linkName"order-to-airline"/>
36            <source linkName"airline-to-agent"/>
37      </invoke>

38      <receive  name=”receiveTickets”,
39                partnerLink="airline"
40                portType="itineraryPT"
41                operation="sendTickets"
42                variable="tickets"
43           <target linkName"airline-to-agent"/>
44      </receive>

45   </flow>

46 </process>
```

The interactions between the partners in the travel agents process are different for the interactions with the customer and the interactions with the airline. In the case of the customer, the customer uses the `sendItinerary` operation on the `itineraryPT` port type provided by the process; this request is then processed by the `<receive>` activity in line 23. No response is being sent back to the customer. In the case of the airline, the process uses the `requestsTickets` operation on the `ticketOrderPT` port type offered by the airline to send a request to the airline (lines 30 to 37). The airline sends its response back by using the `sendTickets` operation on the `itineraryPT` port type, which is processed by the process via the appropriate `<receive>` activity (lines 38 to 44).

3 Partners

As already shown in the travel agent example, business processes that involve Web services often interact with different partners. Partners are connected to a process in a bilateral manner called “partner link type”. A partner link type specifies two port types that are mutually provided and required by the two connected partners; i.e. each partner provides one of the port types. These port types are referred to as “roles”. Here is the definition for the partner link type between the process and the airline:

```
47 <partnerLinkType name="buyerLink">
48    <role name="ticketRequester">
49       <portType name="itineraryPT"/>
50    </role>
51    <role name="ticketService">
52       <portType name="ticketOrderPT"/>
53    </role>
54 </partnerLinkType>
```

The partner link type `buyerLink` consists of two roles. The role `ticketRequester` (line 48 to 50) provides a port of port type `itineraryPT` (line 49), and the role `ticketService` (lines 51 to 53) provides a port of port type `ticketOrderPT` (line 52). The port types are defined somewhere else in appropriate WSDL definitions. When defining a partner within a business process a reference to the partner link type underlying the corresponding bilateral relation between the process and the partner is made (see lines 3 and 7). For example, the `airline` partner link in the travel agent process refers to the `buyerLink` partner link type defined in lines 47 to 54. A partner link definition further specifies which role of the underlying partner link type the process itself accepts ("`myRole`") and which role has to be accepted by the partner ("`partnerRole`"). Accepting a role comes with the obligation to provide the corresponding Web services, i.e. to provide an implementation of the port types of the role. The Web services that are expected by the process from the partner are referenced by the `partnerRole` attribute (e.g. line 10) and the Web services provided by the process and that the partner can rely on and use are referred to by the `myRole` attribute (e.g. line 9).

In other words, the process defines via the `myRole` construct the Web service that represents itself to the outside world; the `partnerRole` construct allows specifying the dependencies of a business process on Web services provided by the outside, i.e. the Web services the business process require and will use.

Multiple partners that implement the same partner link type can be defined in a business process by defining each partner via a separate partner link as shown in the following BPEL fragment.

```
55 <partnerLink name="airline1"
56               partnerLinkType="buyerLink"
57               myRole="ticketRequester"
58               partnerRole="ticketService"/>
59 <partnerLink name="airline2"
60               partnerLinkType="buyerLink"
61               myRole="ticketRequester"
62               partnerRole="ticketService"/>
```

This would allow the travel agent process to communicate with two different airlines at the same time using the same operations and port types.

4 Variables, Properties, and Correlations

Business processes specified via BPEL prescribe the exchange of messages between Web services. These messages are WSDL messages of operations of the port types associated with the roles of the partner links established between the process and its partners. Some of the messages exchanged may be included in the so-called "business context" of the business process. This context is a collection of WSDL messages called "variables" that represent data that is important for the correct execution of the business process, e.g. for routing decisions to be made or for the construction of messages to be sent.

For example, line 27 specifies that the message received from the `customer` via the `sendItinerary` operation of the process' `itineraryPT` port type has to be copied to the `itinerary` variable. And line 34 specifies that the message sent to the `airline`'s `ticketOrderPT` port type as input of the `requestTickets` operation stems from the `itinerary` variable.

Often, the business context is stored persistently to avoid loss of the context, thus, ensuring the correct execution of a business process even in case of planned or unplanned system outages. As the likelihood of such outages increases with the lifetime of a business process, and business processes are typically lasting for long time periods, it is a good practice to make the context persistent.

When messages are exchanged between business partners they typically carry some data that is used to correlate a message with the appropriate business process. For example, the `ticketsMessage` may carry an `orderNumber` that is used by the travel agent and the airline to identify the purchase of tickets for a submitted itinerary of a specific customer; that means it allows the travel agent and the airline to correlate a received message with a particular business process. This kind of correlation data is referred to as "property" in BPEL. Very often the same property is used within different messages as data to be used for correlation. For this purpose, BPEL supports the definition of properties as separate entities. The following BPEL fragment defines the `orderNumber` as a property:

```
63  <property name="orderNumber" type="xsd:int"/>
```

Because a property is used by different messages as correlation data, a mechanism is needed that allows identifying the appropriate field within the message that represents this property. In BPEL, this mechanism is called "aliasing". The following example shows how the `orderNumber` property (line 64) is defined to be the `orderID` field of the `orderInfo` part of the `ticketsMessage`.

```
64  <propertyAlias propertyName="orderNumber"
65                 messageType="ticketsMessage"
66                 part="orderInfo"
67                 query="/orderID"/>
```

5 Activities

Activities are the actions that are being carried out within a business process. The travel agent process already showed some of the activities that can be used within a business process, such as <receive>, <invoke>, or <flow>.

An important action in a business process is to simply wait for a message to be received from a partner. This kind of action is specified via a <receive> activity. It identifies the partner from which the message is to be received, as well as the port type and operation provided by the process used by the partner to pass the message (lines 23 to 27).

A more powerful mechanism is provided by the <pick> activity. This kind of activity specifies a whole set of messages that can be received from the same or different partners. Whenever one of the specified messages is received, the <pick> activity is completed, and processing of the business process continues. Additionally, one may specify that processing should continue if no message is received in a given time. The following BPEL snippet replaces the <receive> activity (lines 38 to 44) that waits for the response from the airline with a <pick> activity.

```
68 <pick>
69     <onMessage partnerLink="buyerLink"
70                portType="itineraryPT"
71                operation="sendTickets"
72                variable="tickets">
73        <empty/>
74     </onMessage>
75     <onAlarm for="P1DT">
76        <invoke partnerLink="customer"
77                portType="travelPT"
78                operation="answerRequest"
79                variable="unableToHonorRequest"/>
80     </onAlarm>
81     <target linkName="airline-to-agent"/>
82 </pick>
```

The <onMessage> element (line 69) is used to define receiving a particular message from a partner via a port type and operation that the process provides. Thus the structure of the <onMessage> specification is the same as for a <receive> activity; the only difference is the mandatory specification of an enclosed activity hat is being carried out when the message has been received. As there is nothing to do when the airline responds, the <empty> activity has been chosen.

The <onAlarm> element (line 75) is used to specify that the activity should wait for some time or until a specified period in time has been reached. If none of the specified messages has been received when the alarm goes off, the enclosed activity is being carried out. The example specifies that the alarm should go off one day after the <pick> activity has started. If this happens, the customer is informed, that the travel agent is unable to handle the customer request.

The start activities of a business process must be <receive> or <pick> activities. Flagging them with createInstance="yes" (lines 87 and 93) indicates that an instance of the specified business process should be created if none exists already. The following illustrates this behavior using a business process that needs to accept the requests from two different partners. The sequence in which the appropriate messages arrive is unclear.

```
83  <receive partnerLink="hotel",
84           portType="roomPT",
85           operation="sendBooking",
86           variable="stayInfo"
87           createInstance="yes"/>
88
89  <receive partnerLink="rentalCar",
90           portType="carPT",
91           operation="sendBooking",
92           variable="rentalInfo"
93           createInstance="yes"/>
```

Regardless which message arrives first, a process instance is created. After the initial message the business process waits for the second one. For example, if the first message is received from a hotel partner, a process instance is created and then the business process waits for the message to arrive from a rentalCar partner.

This approach eliminates the need to have explicit life cycle commands, for example a command to create a process instance. Having no explicit life cycle commands makes life very easy for the requestors of Web services that represent business processes: There is no need to know whether a process instance has already been created or not. As a result, requestors can interact with Web services representing business processes as with any other Web service.

As already pointed out earlier, the travel agent process does not send a response back to the customer; however in most practical cases a response must be returned. As illustrated in the following example, the <reply> activity is used to specify a synchronous response to the request corresponding to a <receive> activity.

```
94  <receive partnerLink="customer",
95           portType="itineraryPT",
```

```
96              operation="sendItinerary",
97              variable="itinerary"
98              createInstance="yes"/>
99
100     <reply    partnerLink="customer",
101               portType="travelPT",
102               operation="sendTickets",
103             variable="tickets"/>
```

In this example, the process provides an in-out operation: The input message of this operation is consumed by the `<receive>` activity, and the output message of this operation is produced via the `<reply>` activity.

If the response to the original request is to be sent asynchronously, the response is delivered via the invocation of a Web service provided by the requester. Consequently, the `<invoke>` activity is used within the process that produces the asynchronous response. The original requester will use a `<receive>` activity to consume the response delivered by the `<invoke>` activity.

Furthermore, the `<invoke>` activity can be used within a process to synchronously invoke an in-out operation of a Web service provided by a partner. As shown in the following example, the `<invoke>` activity needs to identify an input as well as an output variable.

```
104     <invoke   partnerLink="airline"
105               portType="ticketOrderPT"
106               operation="requestTickets"
107               inputVariable="itinerary"
108               outputVariable="tickets"/>
```

All activities discussed so far (except `<pick>`), are called "simple activities" indicating that they have no structure and do not allow to enclose other activities. Other simple activities, called "command" activities for obvious reasons, are: `<wait>` that indicates that the business process should wait for a specified time period or until a specified point in time has been reached, `<empty>` which has no action associated and serves as a means to specify that nothing should be done or to synchronize parallel processing within the process, `<terminate>` to indicate the business process should be terminated immediately, `<throw>` to signal the occurrence of an error, `<assign>` to copy fields from variables into other variables, and `<compensate>` to undo the effects of already completed activities (see section 7).

The travel agent process showed the usage of <flow>, one of the two most important structured activities. It allows defining sets of activities (including other flow activities) that are wired together via <link>s, providing for the potential parallel execution of parts of the flow. Each link may be associated with a transition condition, which is a Boolean expression using values in the different variables of the process. When the business process is being carried out, a particular link is being followed when the associated transition condition evaluates to true.

Other structured activities are: <sequence> that causes the enclosed activities to be carried out in the order they are listed, <switch> to have one path selected out of many paths using selection criteria that references values in containers, and <while> that causes the enclosed activities to be carried out as long as the condition associated with the while-activity evaluates to true.

6 Scopes

In the previous section, <flow> has been identified as one of the most important structured activities. The other one is <scope> which allows building groups of activities and assign certain characteristics to the group of activities. There are no limitations to the type of activities that are enclosed in a scope. The process by default is a scope. A scope has the following characteristics:

```
109    <scope variableAccessSerializable="yes|no">
110
111       <variables>
112           ...
113       </variables>
114
115       <faultHandlers>
116           ...
117       </faultHandlers>
118
119       <compensationHandlers>
120           ...
121       </compensationHandlers>
122
123       <eventHandlers>
124           ...
125       </eventHandlers>
126
127       activity
128
129    </scope>
```

The `<variableAccessSerializable>` property controls how two parallel scopes access variables that are defined outside the individual scopes. When set to yes, access to the variables are serialized. This means when the first scope accesses such a variable that is accessed by both scopes, processing of the second scope is suspended until the first scope has completed processing of the last variable that is accessed by both scopes.

Scopes can have their local variables identified via the `<variables>` element. Only activities within the scope have access to those variables. If a variable with the same name exists in an outer scope, the local variable is used when the name of the variable is used inside the scope.

BPEL processes interact with WSDL ports and such ports may send fault messages back to the process. Furthermore, a process itself might detect erroneous situations that result in internal faults. BPEL provides mechanisms that allow trying to recover from such faulty situations. Central to these mechanisms are so-called "fault handlers" that can catch and deal with faults. They are identified via the `<faultHandlers>` element. A more detailed discussion is provided in the following section 7.

In the process of correcting faults, previously completed activities or set of activities need to be undone. This is the purpose of compensation handlers identified via the `<compensationHandlers>` element. A compensation handler can contain any kind of activity (simple or structured).

When BPEL processes are being carried out, the individual activities interact with partners only at appropriately defined activities. However, in many cases it is important that requests from partners can be accepted at any time or when attached to a scope just as long as the process is running within the scope. This is defined by establishing event handlers, identified via the `<eventHandlers>` element. Event handlers are further discussed in section 8.

As identified via activity in line 127, a scope can contain a single activity; which may be either simple or structured. If structured, the activity may contain another `<scope>` activity as shown in the following example; thus scopes may be nested.

```
130    <scope>
131       <flow>
132          <scope>
133             ...
134          </scope>
135          ...
136       </flow>
137    </scope>
```

7 Fault and Compensation Handlers

A fault handler (lines 138 to 145) defines the set of faults it attempts to handle via a corresponding set of `<catch>` elements (line 139). Within such an element any kind of activity (simple or structured) may be nested. This activity will be performed when the corresponding fault occurs. In the example below, the fault handler catches a `noSeatsAvailable` fault returned by an airline partner. When this fault occurs a corresponding rejection message is sent to the customer via the nested `<invoke>` activity (lines 140 to 143).

```
138    <faultHandlers>
139       <catch  faultName="noSeatsAvailable">
140          <invoke  partnerLink="customer"
141                   portType="sendItinerary"
142                   operation="sendRejection"
143                   inputVariable="rejection"/>
144       </catch>
145    </faultHandlers>
```

When a fault occurs within a scope, the regular processing within the scope is interrupted and the signaled fault is passed to the catching fault handler. The activity nested within this fault handler tries to correct the situation such that regular processing can continue outside the scope or alternate ways to complete the process can be taken.

All of this might require undoing actions that have already been completed within the scope. For example, if the tickets required for a trip are not available, already made reservations for hotel rooms or rental cars must be canceled. The actions required to undo already completed activities are defined via compensation handlers. That means, a fault handler of a scope may make use of compensation handlers to undo actions performed within this scope. It does so via a `<compensate>` activity. The `<compensate>` activity may reference a particular scope (inside the scope that faulted) which causes the compensation handler of the scope to be carried out. If no scope is specified, the appropriate compensation handlers are invoked in the reverse order of execution of the scopes. If the exceptional situation cannot be corrected, the fault handler will re-throw the fault or signal the occurrence of another fault, which will be finally caught by a fault handler of another enclosing scope.

Thus, BPEL allows via its scope mechanism the definition of sets of activities that can be collectively undone in erroneous situations. I.e. such a set of activities is some sort of unit of work, some sort of transaction: Activities that are performed within a scope either all complete or are all compensated [10]. In contrast to this, the well-known "traditional" transactions (like database transactions) are implemented based on locks, i.e. allocating resources to a particular transaction for the duration of the transaction. This takes for granted that transactions are short-lived units of work such that locks can be release fast. Because BPEL scopes are typically long running locking resources doesn't work in practice but one has to use compensation actions instead. This allows releasing locks once an enclosed activity completes, but one has to run compensation logic to undo already completed actions. The resulting units of work or transactions are referred to as "long running transactions".

Long running transactions in BPEL are centered on scopes, and scopes can be nested. There is an agreement protocol between a scope and its parent scope to determine the outcome of the long running transaction represented by a scope. The corresponding protocol has been described in WS-Transaction [12]. While BPEL long-running transactions are currently assuming that a scope and all its nested scopes are contained within a single process and are hosted by a single BPEL engine, the agreement protocol in WS-Transaction does not assume this. Thus, a future extension of BPEL may support long running transactions that are distributed across processes and even across BPEL engines.

WS-Transaction also specifies protocols for coordinating distributed atomic transactions. A future extension of BPEL may support distributed atomic transactions consisting of activities of a single process or even of different processes.

8 Event Handlers

The purpose of event handlers is to carry out some processing that is not part of the main part of the business process. An event handler is activated when the control flow enters the scope the event handler is attached to or if the event handler is associated with the process, when the process is started. An event handler is deactivated when the control flow leaves the scope the event handler is attached to or when the process finishes in case the event handler is associated with the process.

The event handlers shown in the following example are attached to the process is used to terminate the process if either an appropriate message is being received from the customer or the process is running already for two days.

```
146    <eventHandlers>
147       <onMessage partnerLink="customer"
148            portType="itineraryPT"
149          operation="cancel"
150          variable="cancelMessage">
151       <reply partnerLink="customer"
152            portType="itineraryPT"
```

```
153             operation="cancel"
154             variable="cancelAcceptedMessage"/>
155         <terminate/>
156       </onMessage>
157       <onAlarm for="P1DT">
158          <invoke partnerLink="customer"
159                  portType="travelPT"
160                  operation="request"
161                  variable="unableToHonorRequest"/>
162          <terminate/>
163       </onAlarm>
164     </eventHandlers>
```

The first event handler, identified via the `<onMessage>` element (line 147) is carried out when a cancel request is received from the customer as operation `cancel` on port type `itineraryPT`, which is defined as an in-out operation. When the request is received, the customer is informed via an appropriate `<reply>` operation on the receiving port type and operation indicating that the request has been received and is being processed. The `<terminate>` activity causes the termination of the process.

Such a message-based event handler is being carried out, whenever a message is received. If a message arrives when the event handler is being carried out, a new instance of the event handler is created.

The second event handler, identified via the `<onAlarm>` element (line 157) is carried out when the process has been executing for a day. In this case, the customer is informed that the request can not be processed and the process is terminated.

9 Process-based Applications

Applications created with BPEL are so-called "process-based applications" [13], [14]. This kind of application structure split an application into two strictly separated layers: The top layer, the business process, is written in BPEL and represents the flow logic of the application, whereas the bottom layer, the Web services, represents the function logic of the application.

This structure has several advantages over more conventional approaches: (1) the underlying business process as well as the invoked Web services can be changed without any impact on the other Web services within the application or on the Web services that the business process represents, (2) the application can be developed and tested in two separate stages: the business process is developed and tested independent from the development and test of the individual Web services. This approach provides for great flexibility in changing the application. These advantages have been recognized by the UML community too; especially, mappings from UML to BPEL and corresponding relations to model driven architecture (MDA) are on their way [15].

Applications written in BPEL have another major advantage over conventional approaches as they allow tailoring the ready application to the needs and circumstances of a particular environment without touching the application itself. This is achieved by separating the definition of the partners that a business process deals with from the characteristics of the actually involved partners. Within BPEL, one specifies only the port types and operations the different partners are expected to provide.

When such a business process is being carried out, the information about the actual ports or Web services that a concrete partner chosen provides, need to be available. The information about the Web services or ports is collectively subsumed in BPEL under the notion of a "service reference". Concrete mechanisms of providing service references for the different partners within BPEL have been deliberately left out of the specification (aside from a few exceptions). One of the exceptions deals with the situation that a requestor provides the provider with its own service reference so that the provider can respond back to the requestor.

The typical approach for providing service references is to provide this information when the business process is installed ("deployed") in the form of a deployment descriptor. Assigning a service reference to a partner comes in many flavors. In the simplest approach a partner would be assigned a service reference containing fixed information. When the business process is being carried out, this fixed service reference is used to invoke the Web service. In the most complex case, the deployment information could just point to some mechanism, that when the business process is being carried out, determines the appropriate service reference, and possibly invokes the selected Web service right away. This mechanism could, for example, go to UDDI, get all the detail information about potential service providers, and then based on that information selects the most appropriate service provider.

Applications created based on BPEL are portable between environments supporting BPEL and Web services: The BPEL processes can be executed by any BPEL engine, and during their execution a BPEL engine will interact with the Web services that are discovered based on the deployment information.

Besides using BPEL for specifying executable processes, BPEL can be used for specifying "business protocols". A business protocol specifies the potential sequencing of messages exchanged by one particular partner with its other partners to achieve a business goal. I.e. a business protocol defines the ordering in which a particular partner sends messages to and expects messages from its partners based on actual business context. An example for business protocols is the RosettaNet PIPs (see [16]).

Typically, the messages exchanged result from performing activities within internal business processes. Thus, a business protocol may be perceived as a view on a private business process: Internal details like access to backend systems, complete structure of the messages making up the context, complex data manipulation steps, business rules determining branch selection etc are omitted from such a view.

In BPEL the language for specifying business protocols is a subset of the language used for specifying executable processes. This enables to specify an internal executable process together with its views within the same language. It supports an outside-in approach starting with a view and extending it into an internal process, as well as an inside-out approach starting with an internal process projecting it onto its views.

In general, a business protocol (or view, respectively) is not executable: For example, the messages making up the context may be a simple projection of the real internal context messages, it may not be completely specified how messages are contructed that are sent to a partner, branching conditions may not be defined precisely in terms of the data making up the visible context of the business protocol. This is resulting from the fact that a business protocol hides internal details and complexity by will.

Because a business protocol may neither be executable nor deterministic but still expressed as a process, BPEL refers to it as “abstract process”: It abstracts away complex details of an internal executable process. In this sense, abstract processes may be perceived as simple or easy to comprehend processes. And while an abstract process is not guaranteed to be executable, abstract process can be easily specified in a manner such that they are in fact executable! This allows beginning with simple variants of a process and refining them iteratively into the final complex business process.

Finally, an abstract process may be used to easily specify constraints on the usage patterns of a collection of port types: The port types to be constrained are the port types provided by the abstract process, and the operations that are to be constrained are used within activities of the abstract process.

10 Summary

BPEL supports the specification of a broad spectrum of business processes: From fully executable complex business processes over more simple business protocols to usage constraints of Web services. It provides a long-running transaction model that allows increasing consistency and reliability of Web services applications. Correlation mechanisms are supported that allow identifying stateful instances of business processes based on business properties. Partners and Web services can be dynamically bound based on service references

References

[1] W3C, Web Services Definition Language (WSDL) 1.1. http://www.w3.org/TR/wsd.html

[2] OASIS, Universal Description, Discovery &.Integration. http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm#uddiv3

[3] W3C, SOAP Version 1.2. http://www.w3.org/TR/SOAP12

[4] F. Leymann, Web Services: Distributed Applications without Limits, Proc. BTW'03 (Leipzig, Germany, February 26-28, 2003), Springer 2003.

[5] BEA Systems, IBM Corporation, Microsoft Corporation., SAP AG, Siebel Systems. Business Process Execution Language for Web Services. http://www.ibm.com/developerworks/webservices/library/ws-bpel

[6] F. Leymann, Web Services Flow Language (WSFL 1.0), IBM Corporation. http://www-4.ibm.com/software/solutions/Webservices/pdf/WSFL.pdf

[7] S. Thatte, XLANG, Microsoft Corporation. http://www.gotdotnet.com/team/xml_wsspecs/xlang-c

[8] OASIS Web Services Business Process Execution Language TC. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel.

[9] BPMI.org, Business Process Modeling Notation, http://www.bpmi.org

[10] F. Leymann, Supporting business transactions via partial backward recovery in workflow management systems, Proc. BTW'95 (Dresden, Germany, March 22-24, 1995), Springer 1995.

[11] BEA Systems, IBM Corporation, Microsoft Corporation. Web Services Coordination (WS-Coordination). http://www.ibm.com/developerworks/webservices/library/ws-coor.

[12] BEA Systems, IBM Corporation, Microsoft Corporation. Web Services Transaction (WS-Transaction). http://www.ibm.com/developerworks/webservices/library/ws-transpec.

[13] F. Leymann and D. Roller, Workflow-based applications, IBM Systems Journal Vol.26 No. 1., 1997.

[14] F. Leymann and D. Roller, Production Workflow: Concepts and Techniques, Prentice Hall, 2000.

[15] T. Gardner, UML Modeling of Automated Business Processes with a Mapping to BPEL4WS, http://www.cs.ucl.ac.uk/staff/g.piccinelli/eoows/documents/paper-gardner.pdf

[16] RosettaNet. http://www.rosettanet.org

OMG's XML Metadata Interchange Format XMI

Mario Jeckle
University of Applied Sciences Furtwangen
mario@jeckle.de

Abstract: Interest in interchange of models and and data conforming to models attracts increasing interest in these days. Especially since incremental and iterative development processes enter the mainstream market chained modeling tools are common in many current projects. Additionally, users are challenged by new developments like generative approaches or OMG's Model Driven Architecture.
In order to cope with the existing and always increasing heterogeneity among the tools deployed vendor neutral interchange formats have turned into a desideratum.
There are some techniques to transfer data and/or models available on the market. But none of them combines a fixed standard's-based approach with the inherent flexibility to support arbitrary models in a uniform way.
This paper provides a brief overview of all concepts found in OMG's XML METADATA INTERCHANGE Format XMI which strives to support exchange of existing metamodel data as well as the creation of XMI compliant schemata for new metamodels.

1 Methodological and Conceptual Background

The basic idea of the XML METADATA INTERCHANGE format (abbreviated XMI thereafter) is it to provide a sound methodological framework for serializing instances of arbitrary models. This framework should be rich enough to support models organized on various meta layers and thus be capable of encoding the structure of whole languages represented by their respective metamodel, arbitrary models represented as instances of the metamodel they adhere to as well as values serving as incarnations of a concrete model.
This section provides an introduction into the background of the XMI format by reviewing the basic concepts of a four-layer metamodel architecture which forms the grounding of the dominant META OBJECT FACILITY established as a unifying metamodel of various languages standardized by the OBJECT MANAGEMENT GROUP (OMG), such as the UNIFIED MODELING LANGUAGE (UML). Additionally, the seminal idea of XMI's schema production principles is introduced. This deterministic formalism allows the preservation of the same methodological principles throughout the deployment of XMI on the various meta levels.
The reminder of this paper is structured as follows. First the methodological framework is introduced. It is represented by OMG's standardized four layer metamodel architecture which relates modeling using the UNIFIED MODELING LANGUAGE, metamodeling using the META OBJECT FACILITIES, and stream-based interchange based on XML using the XML METADATA INTERCHANGE format.

Section two introduces XMI's basic principles. This includes in detail the transformation algorithm which defines how to generate XML schema representations of arbitrary (meta) models. Additionally, related XML standards such as XML Namespaces, XML Links, and XML Schema are sketched.
Section three provides some deployment scenarios of the XMI approach. This includes a discussion of XMI's role interchanging complete and incomplete models and metadata.
Within section four a brief survey of available support offered by commercial CASE tools is given. Also the idea of deploying XMI as native storage format is discussed.
The final section summarizes some experiences in deploying XMI in projects in research and industry. In detail, results of applying XMI's schema production rules for generating custom schemata are discussed. Additionally, an outlook to the UML 2.0 compliant future version of XMI is provided.

1.1 Metamodeling and its Architectures

One of the most prevalent challenges in the design process of a model interchange format is the support of model instances of various abstraction levels. Models may occur in different flavors of abstraction ranging from meta metamodels describing aspects of a whole modeling language like its static structure or the process deployed to create models to instantiable *user level* models which abstract concrete data instances. As a result, a real interchange format has to support all these various abstraction levels.

Intuitively, there is no limit of abstraction levels since *abstraction* is a term commonly referred to in order to describe a surjective one-to-one relation from a real entity to its abstracted (since this is a cognitive process executed by humans it highly depends on the modelers's perception of reality) counterpart. As long as a modeler is able to abstract an element further s/he formally adds an additional meta layer to the model hierarchy.

For reasons of uniformity the OMG has decided to limit the layers of consecutively stacked models strictly to four. The resulting architecture is shown in figure 1.

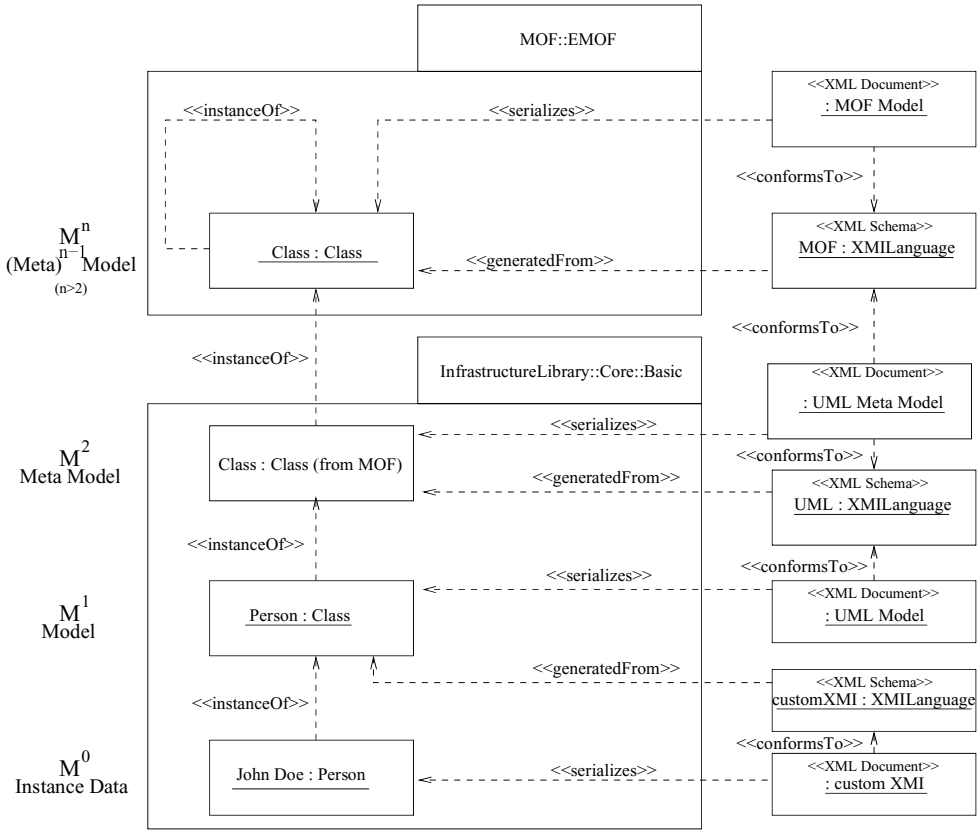


Figure 1: OMG's Four Layer Metamodel Architecture

Restricting the meta layers enables the fixed ascription of an abstraction semantics (i.e. a role adopted by a model layer) to every model within the architecture.
Thus the bottom layer containing concrete data holding objects and association instances representing their interconnections (in UML terms: *links*) is termed the *instance* layer.
Every element of this layer can be directly (more formally: surjectively but not necessarily injectively) mapped onto an element of the *model* layer instantiating the concrete object. This interrelation of the two layers represents the classical *type-instance-dichotomy* [UM01, p. 3-14]. For reasons of brevity, the name of the model layer is shortened to M^1, where the exponent informally denotes the number of occurrences of the letter *m* (which stands for *model*) within the layer's name. Corollary this also clarifies why the instance layer which lacks any occurrences of *m*s is abbreviated as M^0.
The abstraction at the model level, which is normally referred to using the term *class*, both serves as a concrete instance as well as an abstraction of another instance. The relationship between the model level and the *metamodel* level abstraction re-factors the type-instance-dichotomy found at the subjacent architecture layer.
The abstraction is often termed *type*. The connection between a type and its concrete instantiation is represented by an arrow decorated directed edge labeled `instanceOf`, departing from the concrete instance.
This continuous abstraction process is illustrated in figure 1 by introducing `John Doe` as a concrete object typed by the class `Person` which in turn is an instance of the type

entitled `Class` which is predefined by UML's metamodel.
So far, all types and instances are part of a single language framework which is defined but not limited to the UNIFIED MODELING LANGUAGE in figure 1.
By further abstracting the concept *class* as defined by UML, we enter the meta metamodel layer abbreviated to M^3. Since this layer serves as a uniform abstraction of UML and other modeling languages and metamodels (e.g. [IOU+01]) it is addressed by a separate specification titled META OBJECT FACILITY (MOF).

1.2 OMG's Meta Object Facility

The META OBJECT FACILITY [Ob02a] serves as a top level abstraction of various metamodels defined by the OMG. Essentially, MOF is built by abstracting all concepts expressed by different OMG metamodels, e.g. UML, CWM. Furthermore, MOF serves as a termination of an otherwise infinite model stack. This is done by defining all concepts set out by MOF on the concepts defined by the model itself. As a result MOF bootstraps itself and hence terminates the model hierarchy. Formally, MOF is typed by itself as well as it is a valid abstraction of itself.

As shown in figure 1, the class `class` originated by MOF is part of package `MOF::EMOF` which contains MOF's concepts essential for defining metamodels. By definition the prefix *meta* which should precede all concepts defined by the meta metamodel is omitted for brevity [Ob02a, p. 2-5].

Conceptually MOF is organized as a separate model layer which is comprised of the description of the structure and semantics of meta-metadata. Technically, MOF and the metamodel layer use similar concepts. In detail, the essence of the definitions of the concept *class* available in MOF [Ob02a, p. 4] as well as in UML [UM03, p. 6] does not differ. Although the concepts do not match 1:1, the cores of both models are isomorphic and largely share the same concepts w.r.t. the underlying semantics and the naming. This is especially true for MOFs strict subset termed *essential MOF* (EMOF) from which the concept `class` shown in figure 1 originated.

The apparently large conceptual overlap between the metamodel and its metamodel (i.e. the meta metamodel) suggests to consider the organization of the meta metamodel as a subset of the concepts found in the metamodel provided that the metamodel offers enough expressive power to act as a meta metamodel also. On the one hand this would ease the understanding of the metamodel architecture since the concepts found present in the M^3 or higher are identical to concepts found in the modeling language, i.e. the M^2 layer. On the other hand, the selection of a metamodel is crucial. In detail the model has to provide concepts which are mature and semantically rich enough to be propagated to the meta meta-layer without losing expressive power. Otherwise this would brake the abstraction relationship to other metamodels.
Concerning the future development of the four layer metamodel architecture the task force dealing with MOF and UML at the OMG has decided to consolidate and merge the abstract kernel of the UNIFIED MODELING LANGUAGE titled *UML Infrastructure* and the

next generation of the MOF-based meta metamodel by factoring out constructs abstract enough to be re-deployed as meta metamodel from the UML and propagate them to the MOF.
As a consequence of this, the four layer metamodel architecture will be effectively transformed into a hybrid architecture providing four layers of abstraction for all metamodels except for the the seminal UML's one which will reside in a three layer metamodel architecture.

1.3 XML Metadata Interchange

XMI is designed to serve as an interchange format for arbitrary (meta) models. Furthermore, XMI should future proof itself by being capable of also supporting metamodels yet unknown at the time of XMI's definition. Closely related to this is the requirement to offer continuing support by providing compatible encodings for the metamodels changing over time which were initially addressed by the specification. Furthermore, an introduction of interchange formats specific to one version of an existing metamodel would establish a strong dependence of XMI from all metamodel specifications. Changes to these metamodels would in turn imply changes to the interchange format in most cases. Thus XMI would become partly obsolete once a metamodel (specification) is changed.

In order to achieve the highest possible degree of independence the XMI specification does not prescribe concrete schemata supported metamodels. Instead of doing so, XMI sets out a process for automatically generating schemata for arbitrary (meta) models.

OMG's XMI standard defines *schema production rules* which formulate an algorithm to transfer MOF-based (meta) models into XMI-compliant XML grammars which are formulated as instances of the XML SCHEMA vocabulary. The schema production rules can be applied to various models or metamodels as long as they adhere to the structuring principles set out by MOF. As a consequence, the schema production algorithm may be deployed on every model level ranging from pure data models defined as UML class diagrams to meta metamodels. Note that in the context of *XMI* the terms *model* and *metamodel* become interchangeable, likewise do *data* and *metadata*. The various possibilities to create schemata originating from a model are shown in figure 1 by classes connecting to the models by dependency relationships stereotyped with `generatedFrom`.

XMI is MOF-based since it would be virtually impossible to define meaningful production rules that would work on arbitrary models not sharing common characteristics. The approach taken by emphasizing the role of MOF was chosen to provide the commonality among models, allowing the metamodel information to be represented uniformly.

Besides the challenge to define the actual rules driving the process of schema creation which are described by sections 2.3 and 2.2 the schema serialization syntax is crucial to the approach. Since the syntax influences the patterns of interaction with the interchange format and thus promotes or even balks the adoption it could in turn leverage its adoption. The term interaction pattern should refer to the complexity introduced by adding interfaces to existing tools which are capable of importing and exporting the new interchange format.

For reasons of interoperability and market proliferation XMI's group of authors has decided to base the interchange format on the well-known and widely adopted W3C-standardized EXTENSIBLE MARKUP LANGUAGE (XML).
The usage of XML through the XMI specification is two-fold. In essence, the deployment of XML creates the two main constituting parts of XMI.

> **XML DTD/Schema Production Rules**[1] are describing the rules for producing XML Document Type Definitions (DTD) or XML Schema definitions for XMI encoded metadata.[2]
>
> **XML Document Production Rules** are describing the rules for encoding metadata into an XML compatible format.[3]

According to the schemata generated by the schema production principles XMI/XML allows the storage of instances compliant to the metamodel serving as input of schema production. These instances are encoded as XML instance files conforming to the respective XML SCHEMA instance which was derived from the metamodel. By doing so, the XML instances are valid w.r.t. the XML schema which was automatically derived from the metamodel. Thus in turn, provided that the schema production principles governing the generation preserve the structural expressiveness, the XML instances may only store data objects which conform to the metamodel.

1.4 Usage of XMI for Interchange of Business Process Management Data

XMI can be used in two distinct ways to support the interchange of business process management data. First, the XMI format capable of storing arbitrary UML models may be used to encode process management related data which is modeled in a UML compliant manner.
Second, XMI's schema production principles may be applied to arbitrary (MOF compliant) metamodels. This allows the creation of XMI compliant languages for business process management models which are not UML compliant inherently.

Furthermore, provided that a unified business process management metamodel is accepted by an organization XMI may also be deployed to generate an XML vocabulary capable of expressing instances conforming to the business process metamodel. This can be valuable in environments (e.g., a data warehouse storing process descriptions) where process descriptions adhering to diverse metamodels have to be managed

Thus XMI may be used to store business process management data as well as metadata related to these instance data.

[1] All releases of the first version of the specification [UIC 98, UFI 99, Ob02b] use DTDs. This will be changed by the upcoming XMI 2.0 version [Ob03b] which will drop the DTD support in favor of using XML Schema [BLM 01, BM01] for describing structure and content of XMI compliant schemata.

[2] c.f. [Ob02b, p. 1-1]

[3] c.f. [Ob02b, p. 1-1]

2 Deriving XMI Languages from Models

After the introduction of the technical background and the sketch of the basic technical decision to base XMI on XML, this section will provide a more detailed view of the usage of XML techniques by XMI.
Also, an introduction to the basic primitives generated when encoding MOF-based metadata by XMI is given. Finally, the inherent structuring elements present in all schemata which are independent from the source metamodel and generated by deploying XMI's schema production rules are introduced.

2.1 Characteristics of the Language Family

By the use of XML to encode the (meta) models as XML Schema files resp. the (meta) model instances as XML instance files, some elements of the XML technology can be exploited. The reasons for this are two-fold. First, the interoperability of the resulting XML files can be increased by re-using existing approaches and techniques. Second, XML technology provides standardized solutions to classical challenges arising in vocabulary design, such as the unique identification of the vocabulary elements.

In detail, XMI exploits the following XML techniques:

XML Namespaces provide a simple method for qualifying element and attribute names used in XML documents by associating them with namespaces identified by URI references.[4]
Thus, XML namespaces provide the basis for XML-based content syndication and re-use of existing vocabulary and data within new contexts without bearing the danger of conflicting terminology. This also allows usage of XMI fragments within other XML-based languages, but this is beyond the scope of this paper.

XML Links allow elements to be inserted into XML documents in order to create and describe links between resources. They use XML syntax to create structures that can describe links similar to the simple unidirectional hyperlinks of today's HTML, as well as more sophisticated links.[5]
Thus, XML Links provide the basis for avoiding redundancy within the XML serialization of model data by replacing possible error prone redundant occurencies by references to one single occurrence.

XML Schema is a XML schema language[6] which describe the structure and constraints to the contents of XML documents, including those which exploit the XML namespace facility. The schema language, which is itself represented in XML and uses namespaces, substantially reconstructs and considerably extends the capabilities found in XML document type definitions. It also supports definition of dataty-

[4] c.f. [BHL99]
[5] c.f. [DMO00]
[6] Unfortunately, W3C has decided to use the generic name to denote one specific schema language.

pes. These types are to be used in XML Schema as well as other XML specifications. The datatype language, which is itself represented in XML, also provides a superset of the capabilities found in XML document type definitions for specifying data types on elements and attributes.[7]

2.2 XMI Production Rules

Schema production principles are the main cornerstone in striving for independence from the currently addressed (meta) models MOF and UML and in widening the field application to yet unconsidered (and even yet unknown) models. The main ideas for generating XML schemata from static metamodels expressed using MOF's language primitives depicted as UML class diagrams are sketched by this subsection. Afterwards a subset of three basic production rules is discussed. The description of all production rules set out by XMI can be found in the normative documents.

In essence, XMI defines rules to produce XML schema instances from the following basic primitives. This basic primitives are applicable to UML-compliant models as well as to MOF-compliant ones without change.
Note: since XMI version 2.0 will re-factor the production principles this description is based on the available draft [Ob03b] of this upcoming release of the standard. This collection requires some familiarity with the XML techniques and specification introduced in section 2.1.

Class Every class within the metamodel is decomposed into three parts: attributes[8], associations[9], and compositions[10]. A class is represented by an XML element, with an enclosed additional XML element for each attribute, association, and composition. The XML element for the class includes the inherited attributes, associations, and compositions directly since XMI does not utilize XSD's inheritance mechanism.[11]
For example, the representation of the class `person` depicted by figure 1 which does not define any attributes, associations, or containment relationships would be serialized to (simplified form):

```
<xsd:element name="person" type="person"/>
<xsd:complexType name="person">
   <xsd:choice minOccurs="0" maxOccurs="unbounded">
   </xsd:choice>
</xsd:complexType>
```

[7] adapted from [BLM 01, BM01]
[8] i.e., slots for containing values within objects instantiating classes
[9] i.e., associating classes
[10] i.e., a special kind of association narrowing the semantics to a physical containment relationship between a whole and its constituting parts
[11] This was chosen since XSD does solely support single inheritance, but MOF and even UML provide support for multiple inheritance.

Attribute
If attributes present in the metamodel are typed by primitive types or enumerations, then by default XML attributes are declared for them as well as XML elements. The reasons for this encoding choice are including: the values to be exchanged may be very large values and thus unsuitable for XML attributes, and may have poor control of whitespace processing with options which apply only on element content.
Assume the class `person` depicted by figure 1 defines an attribute `name` typed by the primitive type `string`. The simplified schema fragment produced would be as follows:

```
<xsd:element name="person" type="person"/>
<xsd:complexType name="person">
   <xsd:choice>
       <xsd:element name="name" type="xsd:string"
                    nillable="true"/>
    </xsd:choice>
</xsd:complexType>
```

For multi-valued attributes, no XML attributes are declared; each value is encoded as an XML element.

Assume `gender` is an attribute with enumerated values, the type used for the declaration of the XML element and the XML attribute corresponding to the attribute of the class within the metamodel is as follows:

```
<xsd:simpleType name="gender">
   <xsd:restriction base="xsd:string">
       <xsd:enumeration value="male"/>
       <xsd:enumeration value="female"/>
    </xsd:restriction>
</xsd:simpleType>
```

Complex typed attributes, i.e. attributes typed with a class present in the model are handled like associations.

Association Each association is represented in an XML element and/or an XML attribute. The element is declared in the content of the `complexType` for the class that owns the reference. This declaration enables any object to be serialized. The attribute declaration which is also included in the `complexType` declaration serves for referencing the element.
Assume the class `person` depicted by figure 1 defines an association to one or more objects of class `company`. Vice versa, the association is also marked with the multiplicity of `1:*`, i.e. a `company` object may also be linked with one or more objects of the `person`. The simplified schema schema resulting would look like:

```
<xsd:element name="person" type="person"/>
<xsd:complexType name="person">
   <xsd:choice minOccurs="0" maxOccurs="unbounded">
      <xsd:element ref="company"/>
   </xsd:choice>
</xsd:complexType>
```

2.3 Structure of XMI Languages

Besides all variations introduced by schemata derived from arbitrary metamodels XMI's syntactical infrastructure remains the same for all generated schemata which form the *family of XMI languages*.
This section provides a sketch of the infrastructure elements offered by XMI present in all generated models. These elements of the language are the prerequisite of interoperability since they guarantee the structural uniformity of all XMI vocabularies in addition to the uniform production principles.

XMI Header
Every XMI language, regardless of the metamodel it is derived from, defines the fixed start element `XMI`. This XML element serves as root element of all XML elements generated model specific according to the production rules introduced by section 2.2. In order to define a fixed XML schema instance which offers the flexibility to be deployed with schemata generated for arbitrary metamodels, the content model of the element `XMI` is set to `any` with no namespace restriction. This allows, by default, elements of any namespace to be placed as child element of the root element. Additionally, the attribute `processContents` set to `strict` requires the schema processor to validate the embedded content against its respective schema which guarantees the validity of the whole XMI document.

Furthermore, the heading element defines some attributes capable of storing descriptive meta information about the XMI stream serialized. In detail, the concrete metamodel and the version of the XMI specification the XML file is conforming to can be expressed.

```
<xsd:complexType name="XMI">
   <xsd:choice minOccurs="0" maxOccurs="unbounded">
      <xsd:any processContents="strict"/>
    </xsd:choice>
   ...
    <xsd:attribute name="version" type="xsd:string"
                   use="required" fixed="2.0"
                form="qualified"/>
</xsd:complexType>
```

Extension Specification
Opposed to the strictly validated content of the XMI element which constitutes the serialization of the metamodel data the element `extension` allows arbitrary user driven extension be placed within the XMI stream which are serialized purely for tool specific processing.

One use case for introducing these inherently proprietary structures is the intention to store model related data which cannot be covered by instantiating the model's metamodel. Graphical model information like placement and coloring are a well-known example for this since this data cannot be expressed by UML's metamodel prior to version 2.0.

Documentation
Of purely descriptive nature and without any relevance to the encoded metamodel instance are values place inside `documentation` elements within the serialized XMI file. For reasons of identifying the tool which initially created the serialization file the element `documentation` predefines slots for expressing interpreted textual data about the exporter, the concrete version of the tool deployed, and also notices to the human reader of the file.

Linking Attributes
As introduced in section 2.2 every element of a XMI file which represents a metamodel instance can be serialized either *defining* or *referencing* by usage of the predefined linking mechanisms based on XML LINKS.

Every XML element produced from a class within the metamodel is equipped with linking facilities. This is done without introducing redundancy on the meta layer simultaneously which would be the case if multiple definitions of identical linking attributes are generated. Therefore, a set of XML attributes (`LinkAttribs`) is predefined which is referenced from all XML element declarations.

By doing so every XML element produced from a metamodel class offers linking facilities in a uniform way.

```
<xsd:attributeGroup name="LinkAttribs">
   <xsd:attribute name="href" type="xsd:string"
                  use="optional"/>
    <xsd:attribute name="idref" type="xsd:IDREF"
                  use="optional" form="qualified"/>
</xsd:attributeGroup>
```

Identifying attributes
As a prerequisite of general linking XMI defines anchor attributes that links can refer to. These attributes are organized within the re-usage group titled `IdentityAttribs`.

Like the linking attributes discussed before this set of attributes is also referenced by all XML elements produced from classes defined by the metamodel the XMI compliant XML schema is derived from.

```
<xsd:attributeGroup name="IdentityAttribs">
   <xsd:attribute name="label" type="xsd:string"
                  use="optional" form="qualified"/>
    <xsd:attribute name="uuid" type="xsd:string"
                  use="optional" form="qualified"/>
</xsd:attributeGroup>
```

Incomplete Metadata
Besides the transfer of complete XML serializations of metamodel instances (i.e., concrete models), XMI also supports the encoding of arbitrary parts of a model. Serializations of these type are termed transfer of *incomplete meta data* by the specification since the resulting stream does necessarily represent a valid instance of the metamodel.

Serialization of incomplete meta data is therefore not intended to be used by modeling tools for storing models but can be deployed for integrating heterogeneous tools. Since XMI defines three kinds of incomplete meta data this variant of XMI is suitable for coupling tools without relying on a published API.

> **Add** named elements embrace XMI content which is merged with existing content into a new model. The model fragment embraced by `add` has to conform to the schema a.k.a. metamodel of the including content.
>
> **Replace** named elements embrace XMI content which replaces existing content. This is done by replacing model elements by those bearing the same identifying attributes as the ones conveyed in the model fragment.
>
> **Delete** named elements embrace XMI content which deletes existing content from the model. This is done by deleting model elements bearing the same identifying attributes as the ones conveyed in the model fragment.

This three elements could be sent over a network connection as fully legal XML document (e.g., using Web services). CASE tools implementing this will in turn offer an interoperable API for coupling.

3 Deployment Scenarios

This section summarizes usage scenarios which are already common in XMI deployment or which may be promising for the future.

3.1 (Meta) Model Interchange

The native usage of XMI is the Interchange of XMI serialized models. As figure 1 shows, instances of every layer of OMG's four layer metamodel architecture may be expressed

by XMI compliant XML streams. Especially, concerning interchange scenarios involving heterogeneous tool-sets XMI has established itself as the only existing vendor neutral standard providing interoperability.

Furthermore, XMI encoded models are used for tool external storage within revision control systems. As a byproduct of the chosen XML serialization, tools operating on clear text input such as the well-known *diff* may be used to verify consistency of models produced by drawing tools.

3.2 Transfer of Incomplete Model Data

The same is true regarding dynamic tool integration scenarios like those present in incremental and iterative processes as commonly found in agile development. These patterns of development inherently require a tighter level of integration as could be accomplished by integration mechanisms transferring whole models. Since XMI's fragment interchange allows to virtually cut the hot spot of recently changed model elements it is best-suited for this kind of processes.

Moreover, the approach to transfer model fragments instead of complete models conserves bandwidth and avoids whole models be locked by a tool during synchronization.

3.3 Model Driven Architecture

Perhaps the most important application of model interchange in general is formed by the so called MODEL DRIVEN ARCHITECTURE (MDA) which was introduced by OMG in 2001. MDA serves merely as a philosophy and an *umbrella standard* and does not actually introduce new technical ideas by itself. In essence the MDA is made up of a suite of standards including (among others) UML, MOF, and XMI. [Si01, Ob01]

MDA enhances the metamodel architecture depicted by figure 1 by splitting a model residing on a single model layer above the M^0 level into two distinct kinds of models. These models are termed either *platform-specific models* (PSM) or *platform-independent models* (PIM). In terms of MDA PIM denotes a view of a system from the platform independent viewpoint. A PIM exhibits a specified level of platform independence so as to be suitable for use with a number of different platforms of similar type.[12] In contrast, a PSM denotes a view of a system from the platform specific viewpoint. The PSM combines the specifications in a PIM with the details that specify how that system uses a particular platform.[13] In order to achieve this combination within the MDA models of the two kinds mentioned are interconnected by relationships stating that the PSM is generated from a PIM ideally without manual interaction.

Putting MDA into practice requires either modeling tools capable of describing and hand-

[12] c.f. [Ob03a, p. 2-6]
[13] c.f. [Ob03a, p. 2-6]

ling both models and model transformations or tool chains where specific tools interact. Especially the latter requires sufficient interfaces eligible to transfer whole models without loss of information. XMI naturally fits into this scenario since PIM as well as PSM are in most cases expressed using the UNIFIED MODELING LANGUAGE or a customized version.[14] Also, due to the applicability of XMI's schema production rules to arbitrary MOF-based[15] metamodels XMI can serve as a connecting element between any PIM and PSM incarnations.

4 Tool Support

This section briefly summarizes the current state of tools supporting XMI. It should be noted that all tools solely support usage of fixed formats for storing model information. In detail, XMI solely capable of storing UML models. There is currently no tool (neither commercial nor in the academic field) available which fully implements XMI's schema generation production rules.

4.1 Import/Export Interfaces

Most of the UML-supporting CASE and drawing tools currently dominating the market[16] support XMI for exporting UML models and also for importing them back into the tool.

Unfortunately, no overview of the deployment of XMI-based vocabularies produced from MOF-compliant but non UML compliant metamodels can be given since vendors do not provide these information.

In detail both RATIONAL ROSE/XDE[17] (recently acquired by IBM) and TOGETHER[18] (recently acquired by BORLAND) support saving and reading back UML models as XMI-compliant streams. Since the standardization of UML 2.0, which will add capabilities for interchanging the visual representation of a model also, is not finalized yet both tools do not allow to encode diagrams in an interoperable manner. For instance RATIONAL ROSE deploys XMI's extension mechanism sketched in section 2.2 to encode data expressing the graphical placement of model elements.

The general situation shows that approximately 25 percent of the UML supporting tools available on the market currently offer XMI reading and writing capabilities. Up-to-date details of various UML tools and their specific level of support offered for XMI are available at `http://www.jeckle.de/umltools.html`.

[14] The process of customization by extending the predefined modeling primitives is termed *profiling* within the context of UML. Consequentially, the resulting customized UML is termed to be a *UML profile*.

[15] Note: This forms a prerequisite of the applicability of XMI's schema production rules. Hence, only MOF-compliant metamodels can be used a source of the generation process.

[16] A analysis of current market shared can be found in: [Ri02].

[17] `http://www-306.ibm.com/software/awdtools/developer/rosexde/`

[18] `http://www.borland.com/together/index.html`

Slow mass market adoption of XMI may be due to the limitations of XMI documented in section 5.1 and 5.3.

4.2 Native Format Support

Besides the support of XMI by additional filters added to an existing tool XMI may also serve as native internal format. Specifically XML schemata produced from M^2 level metamodels could be deployed in this way. Given a sufficient in-memory representation of XML structures, XMI can also be used as a repository storing all model information at runtime.

Furthermore, data structures for accessing the stored data may be generated directly from the language's metamodel. By doing so a virtual XMI API is established which allows direct operation on the stored model data.

The only commercially available tool currently adhering to this approach is POSEIDON[19] which also offers an early access implementation of the upcoming diagram interchange facilities which will be introduced by UML 2.0's metamodel.

4.3 Meta CASE Tools

In addition to the XMI support by a given tool offering fixed functionalities XMI-based formats can be deployed for encoding custom models which are not directly supported by available tools. A promising application is formed by *meta CASE tools* which are in essence drawing tools enriched by a rule enforcing component.

Instead of hand craft tool support and developing proprietary storage formats for custom languages (i.e. metamodel) meta CASE tools like METAEDIT+[20] could be used. Tools of this type often offer direct support for reading and writing XML compliant data representing metamodel instances (i.e., concrete models) which are stored in the tool's on-line repository.

5 Experiences and Future Work

Finally, we want to provide some documentation of existing experiences and practical problems arising in today's application of XMI. Additionally, an outlook of further developments of the XMI standard is given.

[19] `http://www.gentleware.com`
[20] `http://www.metacase.com`

5.1 XMI's Inherent Heterogeneity Problem

The flexible nature of XMI's schema production rules outlined in section 2.2 inherently introduces an heterogeneity problem concerning the serialization of data instances conforming to a given metamodel. In detail, the schema production rule chosen for the transformation of associations connecting metamodel classes preserves the general net-compliant structure in the inherently strictly hierarchical (i.e. tree-oriented) structures of XML schema.

On the one hand, this offers an XMI export filter the possibility to traverse the metamodel instances in an arbitrary manner provided that it is compliant to the metamodel.
On the other hand, even if the resulting stream of serialized instances is compliant to the XML schema describing the metamodel instances, it may differ vastly from other serializations of the same model.

It is clearly a point of hindrance in market adoption that the OMG standard does not define or even recommend a default encoding. The notion of *default encoding* should refer to predefined paths for fully traversing a given networked model. Due to the lack of these predefined paths any traversal are standard compliant but lead to fairly distinct lexical results. Such an encoding style should include recommended paths which should be followed to access model data.
Also complete abandonment of serialized hierarchical structures would be a valid option for the benefit of interoperability. A flattened structure could serve as equivalent alternative. This would not lower expressiveness since all structures could still be expressed by using XMI's predefined linking mechanisms outlined in section 2.3.

5.2 Schema Production for Custom Languages

Deployment of XMI's schema production rules is currently not widely adopted but as evidence [Je01b] has shown implementation of the schema production algorithm is feasible.

Additionally, industry projects [Je01a] successfully demonstrated the usage of MOF-compliant metamodels generate XMI-compliant XML representations for data interchange.

5.3 Interchange of Visual Models

The revised metamodel introduced by UML 2.0 will add support for storing data describing the visual representation of a metamodel instance, i.e. a UML model. Consequentially, the altered metamodel will result in an update to the XMI serialization of UML created by the schema production rules described in section 2.2. In detail, also the parts of the metamodel which describe the structure of the visual model representation are subject to production rules just like any other metamodel entity.

Besides, the obvious capability of interchanging model data combined with its visual re-

presentation the inclusion of graphical meta data offers to process the visual information separately. An example of such an approach is the generation of Web presentable vector graphics directly from UML models described by [BJMF02].

References

[BHL99] Bray, T., Hollander, D., und Layman, A. (Hrsg.): *Namespaces in XML*. World Wide Web Consortium. Boston, USA. Januar 1999. W3C Recommendation. URL: `www.w3.org/TR/REC-xml-names`.

[BJMF02] Boger, M., Jeckle, M., Müller, S., und Fransson, J.: Diagram Interchange for UML. In: Jézéquel, J., Hußmann, H., und Cook, S. (Hrsg.), *Proceedings of the 5th International Conference on the Unified Modeling Language*. S. 398–411. Oktober 2002. Lecture Notes in Computer Science 2460.

[BLM 01] Beech, D., Lawrence, S., Maloney, M., Mendelsohn, N., und Thompson, H. S. (Hrsg.): *XML Schema Part 1: Structures*. World Wide Web Consortium. Boston, USA. 2001. W3C Recommendation. URL: `http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/`.

[BM01] Biron, P. V. und Malhotra, A. (Hrsg.): *XML Schema Part 2: Datatypes*. World Wide Web Consortium. Boston, USA. 2001. W3C Recommendation. URL: `http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/`.

[DMO00] DeRose, S., Maler, E., und Orchard, D. (Hrsg.): *XML Linking Language (XLink) Version 1.0*. World Wide Web Consortium. Boston, USA. 2000. W3C Recommendation. URL: `http://www.w3.org/TR/2001/REC-xlink-20010627/`.

[IOU 01] International Business Machines Corporation, Oracle Corporation, Unisys Corporation, UBS AG, NCR Corporation, Genesis Development Corporation, Hyperion Solutions Corporation, und Dimension EDI: *Common Warehouse Metamodel (CWM) Specification*. Object Management Group. Framingham, USA. 2001. OMG Document ad/2001-02-01.

[Je01a] Jeckle, M.: Practical usage of W3C's XML-Schema and a process for generating schema structures from UML models. In: *Proceedings of the Second International Conference on Advances in Infrastructure for E-Business, Science, and Education on the Internet*. August 2001. Rome, Italy, 6.-11. August, 2001.

[Je01b] Jeckle, M.: Using XSLT to derive schemata from UML. In: Rahtz, S. und Pawson, D. (Hrsg.), *Proceedings of the International Conference on XSLT*. S. 51–102. April 2001. Oxford, UK.

[Ob01] Object Management Group (Hrsg.): *Model Driven Architecture*. Framingham, USA. Juli 2001. OMG Document ormsc/2001-07-01.

[Ob02a] Object Management Group (Hrsg.): *Meta Object Facility (MOF) Specification*. Object Management Group. Framingham, MA, USA. April 2002. Version 1.4.

[Ob02b] Object Management Group (Hrsg.): *XML Metadata Interchange (XMI)*. Object Management Group. Framingham, MA, USA. Januar 2002. Version 1.2.

[Ob03a] Object Management Group (Hrsg.): *MDA Guide*. Object Management Group. Framingham, USA. 2003. Version 1.0.1, OMG docuemnt omg/2003-06-01.

[Ob03b] Object Management Group (Hrsg.): *XML Metadata Interchange (XMI) Specification*. Object Management Group. 2003. OMG Document formal/2003-05-02.

[Ri02] Rikki Kirzner: *Worldwide Analysis, Modeling, and Design Tools Forecast and Analysis 2002–2006*. IDC. 2002. IDC document no. 24809.

[Si01] Siegel, J. Developing in OMG's Model-Driven Architecture. Object Management Group. November 2001. White Paper, Revision 2.6.

[UFI 99] Unisys Corporation, Fujitsu, International Business Machines Corporation, Softeam, Cooperative Research Centre for Distributed Systems Technology, Recerca Informatica, Oracle Corporation, DaimlerChrysler AG, und Platinum Technology, Inc.: *XML Metadata Interchange (XMI) Version 1.1*. Object Management Group. 1999. OMG Document ad/1999-10-02.

[UIC 98] Unisys Corporation, IBM, Cooperative Research Centre for Distributed Systems Technology, Oracle Corporation, Platinum Technology, Inc., Fujitsu, Softeam, Recerca Informatica, und DaimlerChrysler: *XML Metadata Interchange (XMI). Proposal to the OMG OA&DTF RFP 3: Stream-based Model Interchange Format (SMIF)*. Object Management Group. Framingham, MA, USA. 1998. OMG Dokument ad/98-10-05.

[UM01] UML Partners: *OMG Unified Modeling Language Specification*. Framingham, MA, USA. Februar 2001. Version 1.4, OMG Dokument ad/2001-02-14.

[UM03] UML 2 Partners: *Unified Modeling Language: Superstructure*. Object Management Group. Framingham, MA, USA. April 2003. 3rd Revised Submission. OMG Dokument ad/2003-04-01.

Using the Petri Net Markup Language for Exchanging Business Processes – Potential and Limitations –

Ekkart Kindler

Software Engineering Group
Computer Science Department
University of Paderborn
D-33095 Paderborn
Germany
kindler@upb.de

Abstract The *Petri Net Markup Language* (*PNML*) is an XML-based interchange format for Petri nets. Its focus is on universality and flexibility, which is achieved by a technique for defining new Petri net types through *Petri Net Type Definitions* (*PNTDs*).

Many business process modelling techniques are based on Petri nets. Since PNML provides a means for defining Petri net types, it might be a worthwhile project to define a PNTD for business process models, which supports the exchange of business process models among different BPM tools.

In this paper, we discuss the potential and the limitations of PNML for exchanging business processes. Moreover, we discuss some lessons learned from the standardization of PNML, which could be helpful for the standardization of interchange formats for business process models in general.

1 Introduction

The *Petri Net Markup Language* (*PNML*) is a widely accepted XML-based interchange format for Petri nets [JKW00a, WK03b, BCvH+03], which is currently standardized as ISO/IEC 15909-2. The main problem with devising a standard interchange format for Petri nets is the multitude of existing versions and variants of Petri nets – almost every tool uses a slightly different version of Petri nets, and new versions and variants of Petri nets are invented every year. In order to tackle this problem, PNML provides interfaces for defining new features and new types of Petri nets: the *Features Definition Interface* and the *Type Definition Interface*.

There is an even greater variety of formalisms and notations for business process modelling. Therefore, devising an interchange format for business process models is a much more challenging task. This task, however, is beyond the scope of this paper. But, there are many notations for business models that are based on Petri nets or have an underlying Petri net semantics. Thus, it might be a worthwhile task to devise a Petri Net Type Definition (PNTD) for PNML that supports the exchange of Petri net based business process models.

In this paper, we outline how such a PNTD for PNML could look like[1] and how the corresponding interchange format could be used, and we discuss the potential and the limitations of such an approach. In addition, we will summarize our experiences and the lessons learned during the standardization of PNML, which could be helpful for the standardization of exchange formats for business process models.

The paper is structured as follows: We first give an overview on the principles and the concepts of PNML and, in particular, present its meta model, its XML-syntax, and its Type Definition Interface. Then, we will discuss how PNML could be used for exchanging business processes. In the end, we discuss the lessons learned during the (ongoing) standardization of PNML.

2 PNML

As mentioned in the introduction, PNML was designed to support all kinds, versions, and variants of Petri nets. In order to achieve this flexibility, PNML is split into several parts and is equipped with interfaces for defining new features and new types of Petri nets.

2.1 Overview

The different parts of PNML and their relationships are shown in Fig. 1. The *meta model* defines the basic structure of a PNML file; the *Type Definition Interface* allows the definition of new Petri net types that restrict the legal files of the meta model; and the *Feature Definition Interface* allows the definition of new features for Petri nets. These three parts are fixed once and for all. Another part of PNML, the *Conventions Document*, evolves. It contains the definition of a set of standard features of Petri nets, which are defined according to the feature definition interface. Moreover, there will be several *Standard Petri Net Types*, using some features from the Conventions Document and possibly others. New features and new types can be added to the Conventions Document and to the standard types when they are of common interest. Due to their evolving nature, these documents are best published

[1] We will not present a concrete definition of this format here in order to avoid the publication of a premature definition, which could easily result in a failure of the complete idea (see Sect. 4.3 for details).

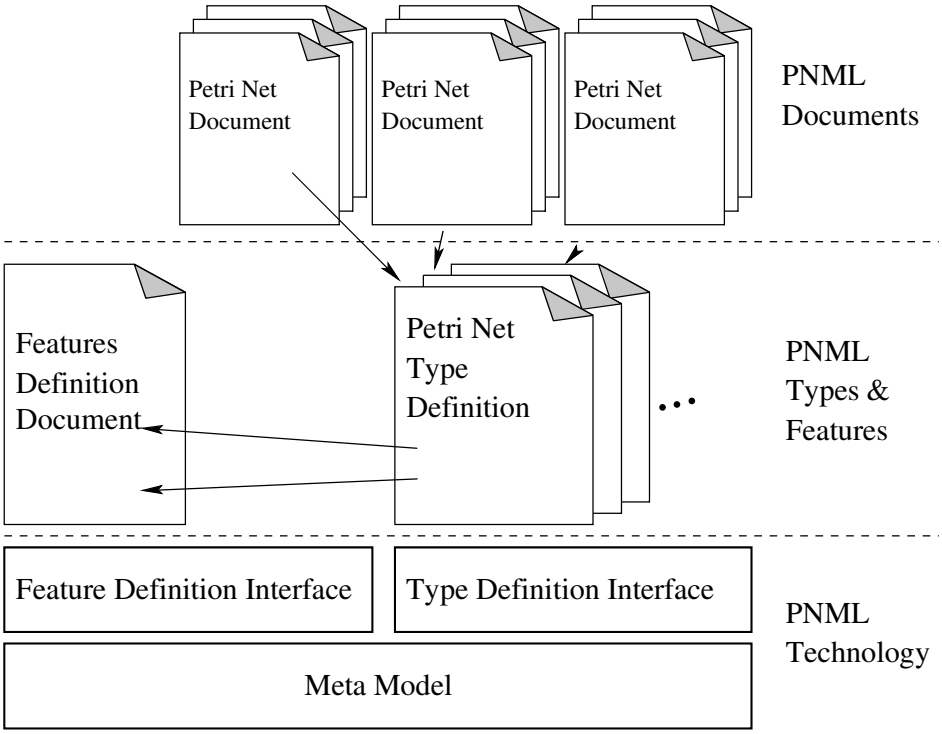


Figure 1: Overview of the parts of PNML

and maintained via a web site. Up to now, only the technological aspects of these documents are defined. The precise process of maintaining these documents, and when and how to update them is not yet fixed. But, it will be in close coordination with the Steering Committee of the annual International Conference on the Application and Theory of Petri nets, which is the major scientific event in the field of Petri nets.

2.2 Meta model

Figure 2 shows that part of the meta model of PNML in UML notation, which is relevant in our context. For the full meta model, we refer to [BCvH+03]. We will explain the meta model below.

2.2.1 Petri nets and objects

A document that meets the requirements of PNML is called a *Petri net document*; it may contain several *Petri nets*. Each Petri net consists of *objects*, which, basically, represent the graph structure of the Petri net. Each object within a Petri net document has a unique *identifier*, which can be used to refer to this object. In

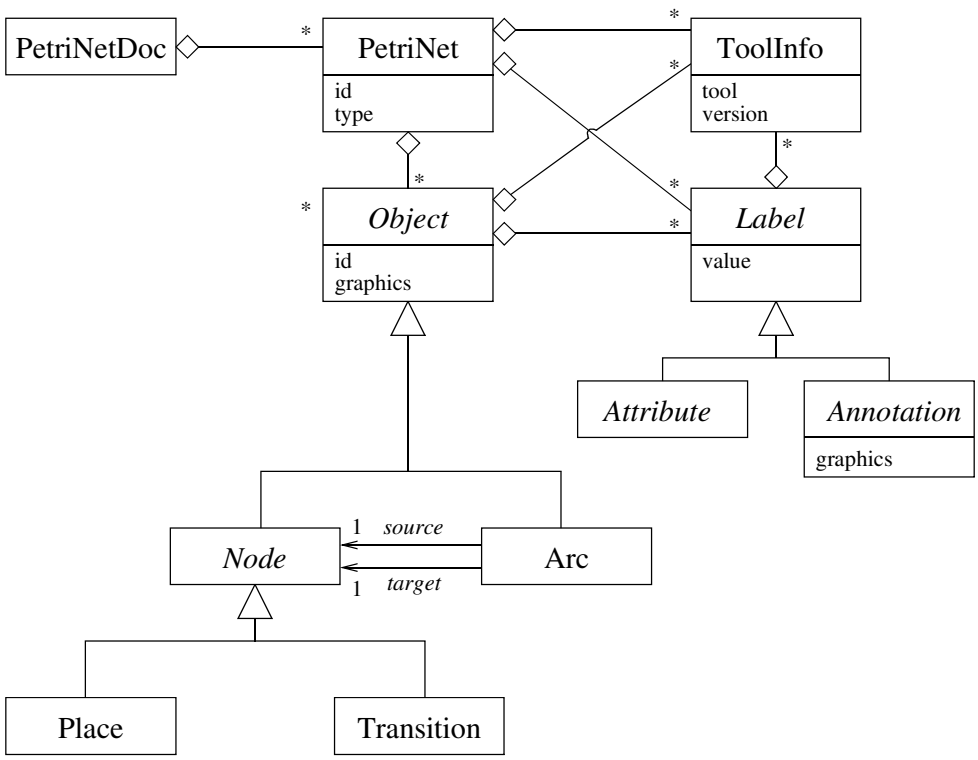


Figure 2: The PNML meta model

basic PNML [2] an object is a *place*, a *transition* or an *arc*. For convenience, a place or a transition is called a *node*.

2.2.2 Labels

In order to assign further meaning to an object, each object may have *labels*. Typically, a label represents the name of a node, the initial marking of a place, the guard of a transition, or the inscription of an arc. In addition, the Petri net itself may have some labels. For example, the declarations of functions and variables that are used in the arc inscriptions could be labels of a high-level Petri net. The legal labels and the legal combinations of labels are defined by the Petri net type. The type of a Petri net is defined by a reference to a unique *Petri Net Type Definition* (PNTD), which will be discussed in Sect. 2.4.

It turned out that two kinds of labels should be distinguished: *annotations* and *attributes*. But this distinction is not relevant for this paper, so we do not discuss this issue here (see [WK03b, BCvH+03] for details).

[2] Basic PNML refers to the version of PNML without any structuring mechanism, which is the version discussed here. In more evolved versions, a net may consist of pages and modules, too.

2.2.3 Graphical information

Each object and each annotation is equipped with graphical information. For a node, this information includes its position; for an arc, it includes a list of positions that define intermediate points of the arc; for an annotation, it includes its relative position with respect to the corresponding object. Additionally, there can be information concerning the size, colour, and shape of a node or an arc, or concerning the colour, font, and font size of a label. Another graphical information can be an image for displaying a node.

2.2.4 Tool specific information

For some tools, it might be necessary to store tool specific information, which is not supposed to be used by other tools. In order to store this information, each object and each label may be equipped with *tool specific information*. Its format depends on the tool and is not specified by PNML. PNML provides a mechanism for clearly marking tool specific information along with the name and the version of the tool that added this information. Therefore, other tools can easily ignore it, and adding tool specific information will never compromise a PNML file.

2.2.5 Pages and modules

More advanced versions of PNML provide mechanisms for structuring Petri nets. In *structured PNML*, a single Petri net can be drawn on several *pages*. In *modular PNML*, there is a concept for defining and instantiating modules, which can be used for constructing systems in a modular and hierarchical way. For more details on structured and modular PNML, we refer to [WK03b, BCvH$^+$03]. The interesting aspect of the page and module concept is that they are completely independent of the Petri net type. They can be used with any type. This way, PNML implements a module concept for any version of Petri nets – even when these versions do not have these concept on their own.

2.3 XML representation

The PNML meta model is translated into XML syntax in a straightforward manner. Technically, the syntax of PNML is defined by a RELAX NG grammar [Relax], which can be found on the PNML web site [PNML].

Class	XML element	XML Attributes
PetriNetDoc	`<pnml>`	
PetriNet	`<net>`	`id`: ID
		`type`: anyURI
Place	`<place>`	`id`: ID
Transition	`<transition>`	`id`: ID
Arc	`<arc>`	`id`: ID
		`source`: IDRef (Node)
		`target`: IDRef (Node)
ToolInfo	`<toolspecific>`	`tool`: string
		`version`: string
Graphics	`<graphics>`	

Table 1: Translation of the PNML meta model into PNML elements

2.3.1 PNML elements

Here, we present the XML syntax in a more compact way: Basically, each concrete class[3] of the PNML meta model is translated into an XML element. This translation along with the attributes and their data types is given in Tab. 1. These XML elements are the *keywords* of PNML and are called *PNML elements* for short. For each PNML element, the aggregations of the meta model define in which elements it may occur as a child element. Note that we have omitted the class Graphics from the meta model in Fig. 2 so as not to clutter the diagram. The classes with associated graphical information are instead indicated by an attribute "graphics".

The data type ID in Tab. 1 defines the declaration of an identifier, which must be unique within the PNML file. The data type IDRef defines references to these identifiers.

2.3.2 Labels

There are no PNML elements for labels because the meta model does not define any concrete label. Concrete labels are defined by the Petri net types. An XML element that is not defined in the meta model (i. e. not occurring in Tab. 1) is considered as a label of the PNML element in which it occurs. For example, an `<initialMarking>` element could be a label for a place, which represents its initial marking. Likewise `<name>` could represent the name of an object, and `<inscription>` could represent an arc inscription. A legal element for a label may consist of further elements. The value of a label appears as a string in a `<text>` element. Furthermore, the value may be represented as an XML tree in a `<structure>` element. An optional PNML `<graphics>` element defines its graphical appearance, and further optional PNML `<toolspecific>` elements may add tool specific information to the label.

[3] A class in a UML diagram is concrete if its name is not displayed in italics.

Parent element class	Sub-elements of <graphics>
Node	<position> (required)
	<dimension>
	<fill>
	<line>
Arc	<position> (zero or more)
	<line>
Annotation	<offset> (required)
	<fill>
	<line>
	<font>

Table 2: Elements in the <graphics> element depending of the parent element

2.3.3 Graphics

PNML elements and labels include graphical information. The structure of the PNML <graphics> element depends on the element in which it appears. Table 2 shows the elements which may occur in the substructure of a <graphics> element. The <position> element defines an absolute position and is required for each node, whereas the <offset> element defines a relative position and is required for each annotation. The other sub-elements of <graphics> are optional. For an arc, the (possibly empty) sequence of <position> elements defines its intermediate points. Each absolute or relative position refers to Cartesian coordinates (x, y). As for most graphical tools, the x-axis runs from left to right and the y-axis from top to bottom. More details on the effect of the graphical features can be found in [BCvH$^+$03].

2.3.4 Example

In order to illustrate the structure of a PNML file, we consider the simple example net shown in Fig. 3. Listing 1 shows the corresponding PNML code.

It is a straightforward translation, where we have labels for the names of objects, for the initial markings, and for arc inscriptions. Note that we assume that the dashed outline of the transition results from the tool specific information <hidden> from an imaginary tool *PN4all*. This appearance is not defined in PNML; it comes from the particular (imaginary) tool.

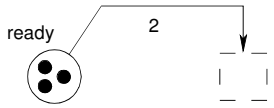


Figure 3: A simple P/T-system

Listing 1: PNML code of the example net in Fig. 3

```
<pnml xmlns="http://www.example.org/pnml">
  <net id="n1" type="http://www.example.org/pnml/PTNet">
    <name>
      <text>An example P/T-net</text>
    </name>
    <place id="p1">
      <graphics>
        <position x="20" y="20"/>
      </graphics>
      <name>
        <text>ready</text>
        <graphics>
          <offset x="-10" y="-8"/>
        </graphics>
      </name>
      <initialMarking>
        <text>3</text>
      </initialMarking>
    </place>
    <transition id="t1">
      <graphics>
        <position x="60" y="20"/>
      </graphics>
      <toolspecific tool="PN4all" version="0.1">
        <hidden/>
      </toolspecific>
    </transition>
    <arc id="a1" source="p1" target="t1">
      <graphics>
        <position x="30" y="5"/>
        <position x="60" y="5"/>
      </graphics>
      <inscription>
        <text>2</text>
        <graphics>
          <offset x="15" y="-2"/>
        </graphics>
      </inscription>
    </arc>
  </net>
</pnml>
```

2.4 Petri Net Type Definition

Next, we discuss the definition of a Petri net type. In order to define a type, we need to define labels first.

2.4.1 Label definition

Listing 2 shows the RELAX NG definition of the label `<initialMarking>`, which represents the initial marking of a place of a P/T-system (cf. List. 1). Its value (in a `<text>` element) should be a natural number, which is formalized by referring to the corresponding data type `nonNegativeInteger` of the data type system of XML Schema [XSch]. Note that the optional graphical and tool specific information do not occur in this label definition; this is not necessary, because these standard elements for annotations in the meta model of PNML are included from the definition of the standard annotation content. Such label definitions can either be given explicitly for each Petri net type, or they can be included in the Conventions Document, such that Petri net type definitions can refer to these definitions.

Listing 2: Label definition

```
<define name="PTMarking"
  xmlns:pnml="http://www.informatik.hu-berlin.de/top/pnml">
  <element name="initialMarking">
    <interleave>
      <element name="text">
        <data type="nonNegativeInteger"
          datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes"/>
      </element>
      <ref name="pnml:StandardAnnotationContent"/>
    </interleave>
  </element>
</define>
```

2.4.2 Petri Net Type Definitions (PNTDs)

Listing 3 shows the Petri Net Type Definition (PNTD) for P/T-Systems as a RELAX NG grammar. Firstly, it includes both the definitions from the meta model of PNML (`pnml.rng`) and the definitions from the Conventions Document (`conv.rng`), which, in particular, contains the definition from List. 2, a similar definition for arc inscriptions of P/T-systems, and a definition for names.

Secondly, the PNTD defines the legal labels for the whole net and the different objects of the net. In our example, the net and the places may have an annotation for names. Furthermore, the places are equipped with an initial marking and the

Listing 3: PNTD for P/T-Systems

```
<grammar ns="http://www.example.org/pnml"
         xmlns="http://relaxng.org/ns/structure/1.0"
         xmlns:conv="http://www.informatik.hu-berlin.de/top/pnml/conv">
  <include href="http://www.informatik.hu-berlin.de/top/pnml/pnml.rng"/>
  <include href="http://www.informatik.hu-berlin.de/top/pnml/conv.rng"/>
  <define name="NetType" combine="replace">
    <text>http://www.example.org/pnml/PTNet</text>
  </define>
  <define name="Net" combine="interleave">
    <optional><ref name="conv:Name"/></optional>
  </define>
  <define name="Place" combine="interleave">
    <interleave>
      <optional><ref name="conv:PTMarking"/></optional>
      <optional><ref name="conv:Name"/></optional>
    </interleave>
  </define>
  <define name="Arc" combine="interleave">
    <optional><ref name="conv:PTArcInscription"/></optional>
  </define>
</grammar>
```

arcs are equipped with an inscription. Note that all labels are optional here. The labels are associated with the net objects by giving a reference to the corresponding definitions in the Conventions Document. Technically, the definition extends the original definition of the net, places and arcs of the RELAX NG grammar for PNML.

2.5 Related work

Of course, there are XML-based formats and other technologies that could be used for interchanging Petri nets. For example, *XMI* [OMG03] could be used for 'mapping' PNML's meta model to XML, or *GXL* [HWS00] could be used for exchanging Petri nets as a graphs.

Besides the fact that all these formats and technologies were developed concurrently, there are some reasons for having a dedicated format for Petri nets. One reason for not using GXL is that PNML can exploit some features that are specific to Petri nets. For example, this applies to the module concept of PNML, which is very specific to Petri nets. One reason for not using XMI is that PNML should be easily usable without being familiar to and without necessarily using XMI technology.

A more detailed discussion of the principles governing the design of PNML can be found in [JKW00b, BCvH+03, WK03b].

3 Business process modelling

In the previous section, we have introduced the basic principles and concepts of PNML. Next, we will discuss in which way PNML could be used as an interchange format for business processes. To this end, we briefly rephrase our understanding of *business processes* and *business process models*, which resembles that of the Workflow Management Coalition [Hol95, WFM99].

3.1 Business processes and their models

A *business process* consist of a set of activities (or tasks) that are executed in some enterprise or administration according to some rules in order to achieve certain goals. The execution of an activity may require an agent or some resources and some information.

A *business process model* more or less exactly captures the rules according to which a specific class of business processes is executed and, in particular, defines which activities are to be performed, in which order they are to be performed, by which agents and resources they are to be performed, and in which way information or documents are used and propagated among activities.

It turned out that business processes have different aspects that can be modelled independently of each other. The *behavioural aspect* models the order in which the different activities must be executed, the *organizational aspect* models the agents and resources of an enterprise and how they are associated with certain activities, and the *informational aspect* models the information used in the process, how the information is represented, and how the information is propagated among the activities.

Actually, there are even more aspects, in particular, when it comes to the automatic execution of business processes by a *workflow management system*. Also *timing aspects* may be very important for doing performance analysis and business process re-engineering. But, we do not go into the details of these aspects here.

By using the term *business process modelling*, people refer to quite different things. Some refer to the modelling of the behavioural aspect only (which is typical for people from the Petri net community), whereas others refer to all aspects mentioned above; some people even refer to the models executed by a workflow management system – in that case they are usually called *workflow models* [LR99, Hol95].

Likewise, the *degree of rigor* of business process models varies. For some people, they are just informal and even incomplete sketches that help giving a rough understanding of an enterprise's or administration's business processes. For others, they are completely formal such that precise analysis or even execution is possible.

There is no harm in the different notions or perceptions of business process modelling. All of them have their benefits and their uses with respect to certain ob-

jectives. But, when it comes to interchange formats for business process models, we need to be aware of the different perceptions of business process modelling and the different objectives. To us it is not clear at all, whether there can be or there should be a single interchange format serving all purposes and perceptions in business process modelling.

3.2 Petri nets

Petri nets are well-known for being a formal and rigorous modelling technique. Therefore, Petri nets are well-suited for rigorous business process modelling. Classical Petri nets such as Place/Transition-Systems (P/T-Systems) cannot distinguish between different kinds of token (there are black tokens only) and there is no means for modelling data. This is the reason why Petri nets are usually used for modelling the behavioural aspect of business processes only. Maybe, the best known approach is the one by van der Aalst, in which he proposes a special kind of unmarked P/T-Systems with a single input place and a single output place, called *workflow nets* [vdA97, vdAvH02]. Sometimes, however, Petri nets are considered to lack modelling power. Van der Aalst himself proposes some extensions called *YAWL* [vdAtH02].

Classical Petri nets are also good for analysis and planning of resource assignment to tasks and for business processes in particular. When using timed or stochastic versions, Petri nets can also be used for doing performance analysis: And these techniques turn out to be quite effective. But, this does not necessarily mean that Petri nets are good for modelling the organizational aspects of business processes. In fact, they are not good at all[4]. When it comes to realistic processes and organizational models, the corresponding Petri net become quite complex and inscrutable. So it would be much better to use a more appropriate notation for modelling the organization and the relation of its resources to the activities. Then we could automatically translate this model to a Petri net, which can be used for automatic analysis. In order to make this approach work, an interchange format for the organization aspect of business processes must have a clean and simple meta model and a mechanism for associating the resources of the organizational model with the activities in the behavioural model. This notation could and should be independent of the particular modelling formalism for the behavioural aspect.

In classical Petri nets, the information aspect of a business process cannot be modelled. But, there are different versions of high-level Petri nets that allow us to model information and data. Actually, the information model could be defined in almost any notation; then, a high-level Petri net could be used for modelling the propagation of the information, resp. data or documents between different activities. Again, (high-level) Petri nets may not be good at modelling the structure

[4]Too often, the distinction between being a good modelling notation and providing a good analysis techniques is not clearly made. This might be the source of many misunderstandings in discussions on appropriate and inappropriate modelling and analysis techniques.

of the data or in defining the information model – most of them use a notation that is not specific to Petri nets at all. But, Petri nets are good at modelling how information is propagated between different activities, once the information model is defined.

3.3 PNML for business process models

As indicated in the previous section already, there are several ways in which PNML could be used as an interchange format for business process models.

3.3.1 Workflow nets

First, PNML could be used as an interchange format for the behavioural aspect of business processes only. To this end, we could define a PNTD for *workflow nets* as defined by van der Aalst [vdAvH02]. This PNTD would essentially be the PNTD for P/T-Systems without the labels representing the initial marking; additionally, there would be a distinguished start and end place.

The corresponding PNTD is very simple. But, we do not give a PNTD for workflow nets, here. With this PNTD, PNML could be used for exchanging workflow nets.

3.3.2 Resources

Of course, we can easily extend the PNTD for workflow nets with some labels that define the needed resources for each activity (resp. the transition representing it). The syntax of these labels, however, will depend on the modelling notation for the organizational model resp. its underlying meta model. Once the meta model resp. the interface for referring to an organization model is fixed, it should be an easy task to define the syntax of the resource labels.

3.3.3 Information propagation

Similarly, we could easily devise a PNTD for high-level Petri nets in order to model the propagation of information between different activities. The concrete syntax of the arc-labels would depend on the meta model resp. its interface of the information aspect.

3.3.4 Extensions

For particular purposes, we could easily extend the format with all necessary features that are available in the Conventions Document of PNML. For example, we could use labels for timed Petri nets or for stochastic Petri nets in order to represent timing information.

3.3.5 Interfaces

In essence, a PNTD for defining business processes would be a definition of workflow nets, equipped with some additional information. For defining the concrete labels for the informational aspect and for the organizational aspect, however, it would be necessary to define interfaces resp. meta models for these aspects first. These interfaces, however, could and should be completely independent of Petri nets or any other model for the behavioural aspect. Therefore, discussing and fixing models for these aspects is more important than devising a PNTD for workflow nets. Once the meta models resp. interfaces for the other aspects are defined, it is straightforward to define a PNTD for the behavioural aspect of business process models.

4 Lessons learned with PNML

The discussion of XML-based interchange formats for Petri nets started about four years ago. And the standardization process is not finished yet; in fact, the official standardization as ISO/IEC 15909-2 has just begun. So, we are still learning some lessons on the standardization of PNML.

Nevertheless, we report on some of the lessons learned to date in the hope that they might be helpful in the standardization of an interchange format for business processes. But, we are aware that the standardization of business processes is a much more difficult issue.

4.1 Organization

The standardization of an XML-based[5] interchange format for Petri nets was started in 2000 with an workshop at the annual International Conference on Application and Theory of Petri Nets [BBK+00]. This and all subsequent activities and events were closely coordinated with the Steering Committee of the International Conference on Application and Theory of Petri Nets. Though there have been concrete proposals for exchange formats for Petri nets at the first workshop, the focus of this workshop and the discussions were on the principles and objectives of such an interchange format and on ideas how to deal with the variety of different Petri net types. At this workshop, we agreed that a standard interchange format would be helpful even if not all information can be interchanged among different tools. A format that would help to interchange the basic structure of a net among different tools would be helpful in many cases.

[5] Actually, there have been other interchange formats for particular versions of Petri nets long before that. For example, there was the *Abstract Petri Net Notation* (*APNN*) [BKK95].

After the first workshop in 2000, we had more or less informal meetings every year during the annual International Conference on Application and Theory of Petri Nets, which were open to everybody. Finally, these meetings resulted in a joint paper on PNML [BCvH+03], which will be the basis for the standard ISO/IEC 15909-2.

The lesson learned from this is that it is important to have a common understanding of the objectives and the purposes of an interchange format before making proposals for such a format.

4.2 Principles

During the first workshop, it became clear that dealing with the variety of different versions and dialects of Petri nets would be one of the main issues in a new standard. Therefore, openness and extensibility of the format were the main principles driving the design of PNML.

PNML provides extensibility in several ways. One way is the features and type concepts, which allows us to define new labels for Petri nets and new Petri net types. It will be an ongoing activity to update and to maintain the standard features and the standard Petri net types. But, everybody is free to define an own Petri net type – other tools will be able to understand at least that part that refers to standard features.

Another way for providing extensibility is tool specific information. Every tool is allowed to store its private information for the different objects within a Petri net. This way, all information that is not covered by PNML can be stored in a PNML file, and tools are not forced to have their private format in addition to PNML. A tool can store any information necessary within a PNML file. On the one hand, this is an important feature for tool providers to support PNML. On the other hand, it is also a dangerous feature: If different tools store important information as tool specific information only, this information cannot be used by other tools anymore. Therefore, we discourage the use of tool specific information wherever possible and we try to make sure that there are standard features and standard Petri net types for all important features and Petri net types available.

A similar problem might occur concerning graphics. Initially, we had only very few graphical features in PNML and considered all other graphical information to be tool specific. But, we realized that this way most graphical information might go to tool specific information and, therefore, would be lost for other tools. Therefore, we now aim at providing a way to represent all graphical features that are well-established in Petri net tools within PNML itself[6]. Only very special or fancy graphical features cannot be expressed in PNML. This way, PNML not only guarantees the same appearance of a Petri net in different tools, we can even provide

[6]In fact, many people are still not aware of the much richer graphical features of the current version of PNML because they had a look to now outdated versions only. This is another argument for not starting with a half-baked definition.

a standard transformation of PNML files to SVG [FJe03]. Actually, we define the graphical appearance of a PNML file by an XSLT transformation to SVG, which was first proposed in [Ste02].

4.3 Tools

Fortunately, many tool providers started to implement PNML, once the first draft of PNML was published. Maybe, this was the most important factor to the success of PNML. But, one problem was that the first papers focused on the concepts of PNML and were incomplete in some technical issues[7]. Therefore, there have been different variants and versions of PNML before we came up with a complete version. Moreover, we had to change the definition of some constructs in order to get a clearer design of PNML. Though these could be considered to be minor changes only, the tools using PNML already ran into problem.

The lesson learned from this is that already the first draft should be worked out in full technical detail and a validation tool should be available right from the beginning. This can help avoiding frustration of tool providers, who are so important for the success of a standard. This is the reason why we do not provide a concrete definition of a PNTD for business process models here.

5 Conclusion

In this paper, we have given an overview on PNML, and we discussed how PNML could be used for exchanging business process models.

The idea boils down to providing a Petri Net Type Definition (PNTD) for the behavioural aspect of business process models. Other aspects could be incorporated into this model too. But, we do not propose to use Petri nets for modelling the informational or the organizational aspect of a business process. Rather, we would like to use some more appropriate formalism, which provides a clear interface such that Petri net models can refer to these models in order to associate resources with activities and to model the propagation of data based on these models.

In a nutshell, we propose to start with a standardization of the interfaces of the meta models for the different aspects of business process models, rather than to start with a format for business processes themselves. Once, these interfaces are defined, devising a PNTD for the behavioural aspect of business processes will be easy.

[7] We must admit that our own tool, the *Petri Net Kernel* [WK03a], implemented PNML slightly different from its specification.

Acknowledgments

Some parts of this papers are taken from the PNML paper [BCvH+03] with only minor modifications. I would like to thank all co-authors of that paper for their permission to use these parts and for their encouragement to carry on. In particular, I would like to thank Michael Weber and Renier Post for their comments and suggestions on an earlier version of this paper. Moreover, I would like to thank the anonymous reviewers for their comments and questions, which helped to improve the presentation of this material.

References

[BBK+00] R. Bastide, J. Billington, E. Kindler, F. Kordon, and K. H. Mortensen, editors. *Meeting on XML/SGML based Interchange Formats for Petri Nets*. University of Aarhus, Department of Computer Science, June 2000.

[BCvH+03] Jonathan Billington, Søren Christensen, Kees van Hee, Ekkart Kindler, Olaf Kummer, Laure Petrucci, Reinier Post, Christian Stehno, and Michael Weber. The Petri Net Markup Language: Concepts, Technology, and Tools. In W. van der Aalst and E. Best, editors, *Application and Theory of Petri Nets 2003, 24^{th} International Conference*, *LNCS* 2679, pages 483–505. Springer, June 2003.

[BKK95] Falko Bause, Peter Kemper, and Pieter Kritzinger. Abstract Petri Net Notation. *Petri Net Newsletter*, 49:9–27, October 1995.

[FJe03] J. Ferraiolo, F. Jun, and D. Jackson (eds.). Scalable Vector Graphics (SVG) 1.1 Specification. URL `http://www.w3.org/TR/SVG11/`, 2003.

[Hol95] David Hollingsworth. The Workflow Reference Model. Technical Report TC00-1003, The Workflow Management Coalition (WfMC), January 1995.

[HWS00] Richard C. Holt, Andreas Winter, and Andy Schürr. GXL: Towards a Standard Exchange Format. In 7^{th} *Working Conference on Reverse Engineering*, pages 162–171. IEEE Computer Society, 2000.

[JKW00a] Matthias Jüngel, Ekkart Kindler, and Michael Weber. Towards a Generic Interchange Format for Petri Nets – Position Paper. In R. Bastide, J. Billington, E. Kindler, F. Kordon, and K. H. Mortensen, editors, *Meeting on XML/SGML based Interchange Formats for Petri Nets*, pages 1–5, June 2000.

[JKW00b] Matthias Jüngel, Ekkart Kindler, and Michael Weber. The Petri Net Markup Language. *Petri Net Newsletter*, 59:24–29, October 2000.

[LR99] Frank Leymann and Dieter Roller. *Production Workflow: Concepts and Techniques*. Prentice-Hall PTR, Upper Saddle River, New Jersey, USA, 1999.

[OMG03] XML Metadata Interchange (XMI) Specification, Version 2.0. Technical Report formal/03-05-02, The Object Management Group, Inc., May 2003.

[PNML] The Petri Net Markup Language. URL `http://www.informatik.hu-berlin.de/top/pnml/`. 2001/07/19.

[Relax] RELAX NG Specification. URL `http://www.oasis-open.org/committees/relax-ng/`. 2001/12/03.

[Ste02] Christian Stehno. Petri Net Markup Language: Implementation and Application. In J. Desel and M. Weske, editors, *Promise 2002*, *Lecture Notes in Informatics* P-21, pages 18–30. Gesellschaft für Informatik, 2002.

[vdA97] W.M.P. van der Aalst. Exploring the Process Dimension of Workflow Management. Computing Science Reports 97/13, Eindhoven University of Technology, September 1997.

[vdAtH02] W.M.P. van der Aalst and A.H.M. ter Hofstede. YAWL: Yet Another Workflow Language. Technical Report QUT Technical report, FIT-TR-2002-06, Queensland University of Technology, Brisbane, 2002.

[vdAvH02] Wil van der Aalst and Kees van Hee. *Workflow Management: Models, Methods, and Systems*. Cooperative Information Systems. The MIT Press, 2002.

[WFM99] Workflow Management Coalition: Terminology & Glossary. Technical Report WFMC-TC-1011, The Workflow Management Coalition (WfMC), February 1999.

[WK03a] Michael Weber and Ekkart Kindler. The Petri Net Kernel. In H. Ehrig, W. Reisig, G. Rozenberg, and H. Weber, editors, *Petri Net Technologies for Modeling Communication Based Systems*, *LNCS* 2472, pages 109–123. Springer, 2003.

[WK03b] Michael Weber and Ekkart Kindler. The Petri Net Markup Language. In H. Ehrig, W. Reisig, G. Rozenberg, and H. Weber, editors, *Petri Net Technologies for Modeling Communication Based Systems*, *LNCS* 2472, pages 124–144. Springer, 2003.

[XSch] XML Schema. URL `http://www.w3.org/XML/Schema`, April 2000. 2002-03-22.

Exchanging EPC Business Process Models with EPML

Jan Mendling

Abteilung für Wirtschaftsinformatik
Wirtschaftsuniversität Wien,
A-1090 Wien
jan.mendling@wu-wien.ac.at

Markus Nüttgens

Universität des Saarlandes
D-66041 Saarbrücken
markus@nuettgens.de

Abstract: In this paper EPML is presented as an interchange format for EPC business process models. EPML builds on EPC syntax related work and is designed to be applicable as a serialisation format for EPC modelling tools. After a description of EPML in the large, examples are given to illustrate selected representational aspects including flat and hierarchical EPCs, business views, and graphical information.

1. Exchanging Business Process Models

Today business process modelling is mainly used in two different contexts: business analysts use process models for documentation purposes, for process optimization and simulation; while information system analysts use them on the middleware tier in order to glue together heterogeneous systems. For both of these layers analysts have a variety of tools to choose from in order to support modelling of processes. In 2002 Gartner Research distinguishes 35 major vendors of such software [Ga02]. Heterogeneity of these tools causes huge interoperability problem in this context. A recent survey of DelphiGroup [De03] identifies the lack of a common and accepted interchange format for business process models as the major detriment for business process management.

Event-Driven Process Chains (EPC) [KNS92] are a wide-spread method for business process modelling. SAP AG has been using them to express their SAP reference model [Ke99]. Motivated by the heterogeneity of business process modelling tools, a proposal for an interchange format for EPCs is in progress of development. It is called EPC Markup Language (EPML) [MN02, MN03b, MN03c]. The establishment of a standardized representation of business process models may be even more beneficial than in other domains of standardization, because it may be used in two different directions: horizontal interchange will simplify the integration of BPM tools of the same scope. Vertical interchange can leverage the integration of simulation engines, execution engines, and monitoring engines [Wf02]. Standardization might be a crucial step to close the engineering gap between business process modelling and implementation.

This paper gives an overview over EPML. Section 2 introduces Event-driven Process Chains as a method to express business process models, their syntactical elements, and related research on EPC syntax. Section 3 presents EPML general design principles and XML design guidelines that have guided the specification. Section 4 explains how the syntax elements of EPML relate to each other and outlines why edge element lists are used to describe EPC process graphs in EPML. The Sections 5 to 8 introduce specific aspects of EPML by giving examples: Section 5 presents a simple EPC example and its EPML syntax representation; Section 6 shows how hierarchies of EPCs are expressed; Section 7 discusses how business perspectives can be included in an EPML file; and Section 8 shows which graphical information can be attached to a process element Section 9 concludes and lists future directions of research.

2. Event-Driven Process Chains (EPCs)

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], 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 aspect has recently been formalized by Kindler [Ki03]. In this paper we will focus on EPC syntax referencing to based on the syntax definition of EPCs in [NR02]. Therefore we give a brief survey of syntax related work.

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. OR-Split and XOR-Split 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 [Ru99] 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".

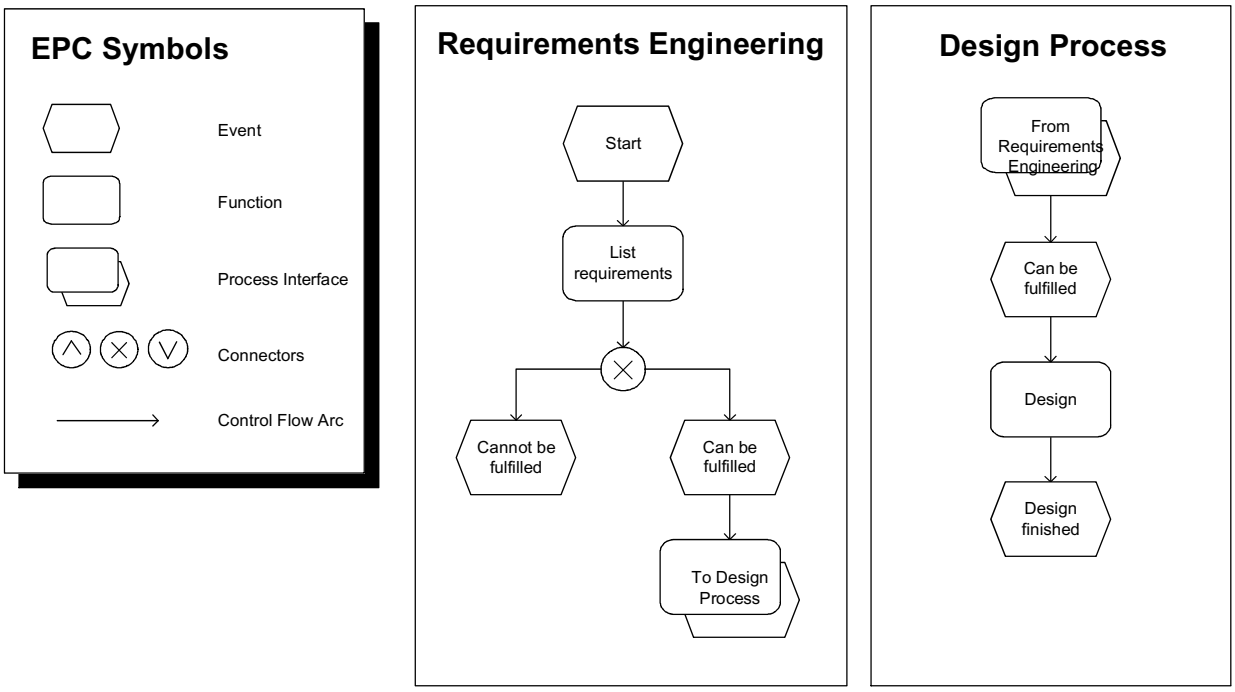


Fig. 1. EPC example of a simple requirements engineering process. The connector represents an "exclusive or". After "Can be fulfilled" a process interface links to the design process.

Work on EPML started off in 2002 mainly inspired by heterogeneity of business process modelling tools and the potential of efficiency gains for the use of an intermediary format [WHB02, MN03b]. As a first step, comparable efforts towards standardized interchange formats in the area of Petri Nets, BPML, and UML have been analyzed [MN02]. Work on syntactical correctness led to a revised EPC syntax definition based on implicit arc types and related syntax properties [MN03a]. An analysis on EPC syntax validation discusses in how far these syntax properties can be expressed via standard XML Schema languages [MN03c]. A proposal for an EPML schema is presented in [MN04] and has been made available at http://wi.wu-wien.ac.at/~mendling/EPML.

3. Design Principles

Towards the definition of an XML syntax for EPC models, the global goal of defining a tool and platform independent XML-based interchange format for EPCs has to be translated into domain-specific design principles in order to derive design decisions. Domain-independent XML design guidelines standardize the way how things are put into XML syntax.

3.1. EPML General Design Principles

In order to put EPML design principles into context, we present design principles proposed for ASC X12 Reference Model for XML Design (X12) [AN02] and Petri Net Markup Language (PNML) [Bi03]. 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 organisations 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 for Petri Nets is governed by the principles flexibility, no ambiguity, and compatibility [Bi03]. *Flexibility* is an important aspect for Petri Nets, because all kinds of currently discussed and also prospective classes of Petri Nets shall be stored. This will be achieved with labels which can be attached to arcs and nodes. *No ambiguity* refers to the problem of standardized labels. Therefore, Petri Net Type Definitions define legal labels for particular net types. *Compatibility* deals with the problem of semantically equivalent labels used by different Petri net types. These overlapping labels shall be exchangeable.

The EPML approach reflects these different design principles. It is governed by the principles of readability, extensibility, tool orientation, and syntactical correctness [MN03b]. *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 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. An important aspect of BPM is to provide different business perspectives and views on a process. EPML should be capable to express arbitrary perspectives instead of only supporting a pre-defined set. Section 6 is dedicated to this issue. *Tool orientation* deals with graphical representation of EPCs. This is a crucial feature, because BPM tools provide a GUI for developing models. EPML should be able to store various layout and position information for EPC elements. Finally, *syntactical correctness* summarizes aspects dealing with EPC syntax elements and their interrelation. The following paragraph will discuss general XML design aspects.

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), multi-value dependency (MVD), and join dependency (JD) have been formally described [Bi95]. 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, a normal form has been presented for XML, called (XNF) [EM01, AL02]. In [AL03] 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 [AL03]. 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, an XML schema subject to FDs is well-designed iff it is in XNF [AL03]. A violation of XNF implies redundancies in that 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 that moves attributes and creates new elements until XNF is achieved [AL03]. For XML reference model design this implies that there should be no XPath [CD99] 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. Documents from ISO [ISO01], SWIFT [SW01], and X12 [AN02] 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, MISMO, 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 MISMO, 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 [AN02]. 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 [Be01, BM01]. Further, attributes are considered to provide a better readability of content [AN02]. 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 [AN02] 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. EPML in the Large

<epml> is the root element of an EPML file. Like all other elements it may have <documentation> or <toolInfo> child elements. These may contain data that has been added by the editor of the EPML file or tool specific data attached by an application. These two elements are of XML Schema type anyType which means that they may hold arbitrary nesting of XML data. It is recommended to use only standardised Dublin Core Metadata Elements [DC03] for documentation of the EPML file, and 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 presentation of a model. General graphic settings may be defined in the <graphicsDefault> 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 MS Visio [Mi03] 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 [NR02] 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. The detailed discussion of flat EPC Schemas is left to the following Section; here, it is sufficient to have a general understanding of what EPCs are. Syntactically, a hierarchy relation connects functions or process interfaces with other EPC processes. Semantically, it refers to the call of sub-processes. <epml> also has a <definitions> child element which is explained in conjunction with the <directory> element. The <view> element allows business views and perspectives to be defined. Its <unit> element is a container for information about an entity which is important in a business process. This unit may be attached to control flow elements. Figure 2 gives an overview over EPML via a metamodel; Table 1 illustrates the content model of high-level EPML elements.

In EPML a hierarchy of processes is organised by the help of directories. A <directory> holds a @name attribute, other directories, and/or EPC models. Each <epc> 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. The EPC control flow elements will be discussed in paragraph 4.2. In a hierarchy of EPC models there may be the problem of 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 precisely the aim of the <definitions> element. It serves as a container for control flow elements that are used more than once in the model hierarchy.

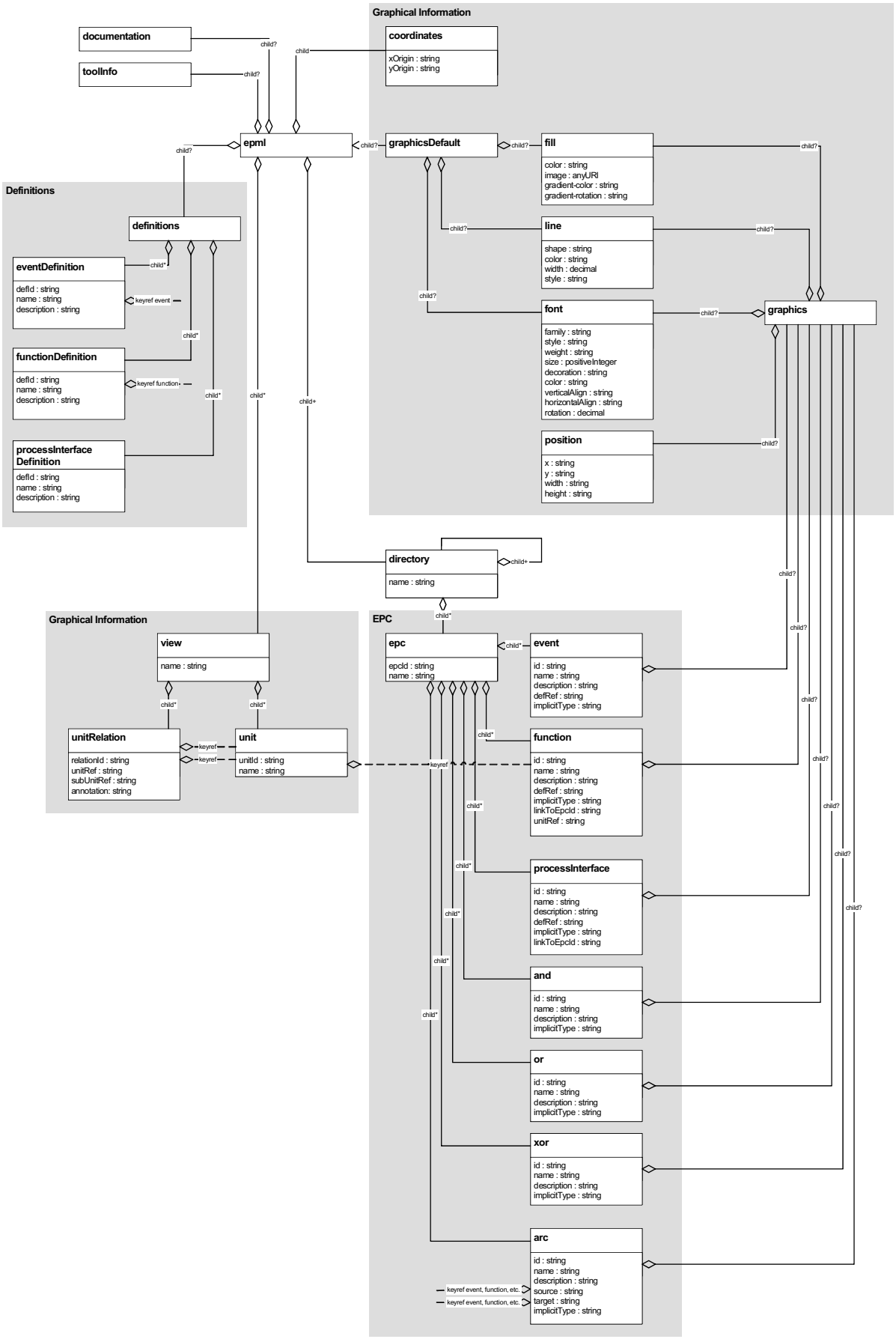


Fig. 2: Overview over EPML including its main syntax elements.

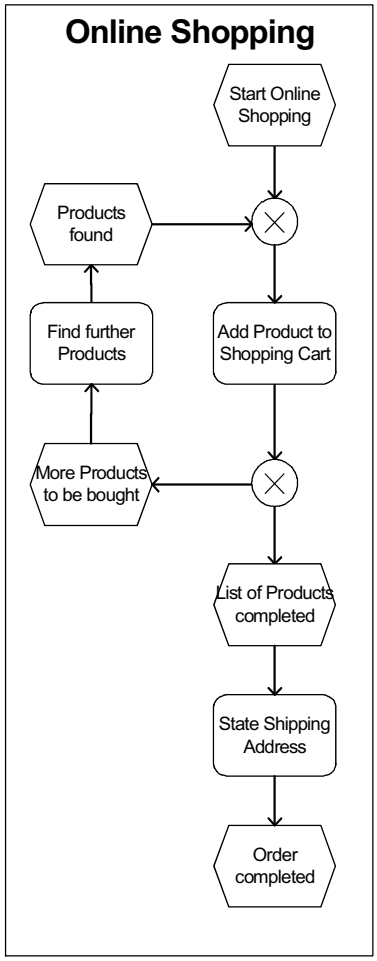
EPML element	Attributes and Sub-Elements
<epml>	<documentation> ? <toolInfo> ? <graphicsDefault> ? <coordinates> <definitions> <view> * <directory> +
<definitions>	<documentation> ? <toolInfo> ? <eventDefinition> * <functionDefinition> * <processInterfaceDefinition> *
<directory>	@name <documentation> ? <toolInfo> ? <directory> * <epc> *
<epc>	@epcId, @name <documentation> ? <toolInfo> ? <event> * <function> * <processInterface> * <and>, <or>, <xor> * <arc>

Table 1: High level elements of an EPML file.

5. 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. After starting the process, a product is added to the shopping cart. When the buyer wants to buy more, he adds another product to the shopping cart until the list of products is completed. He then completes the order by stating her shipping address. The code on the right hand side of Fig. 3 shows an excerpt from an EPML file corresponding to that process. The root tag of every EPML file is <epml> and it must belong to the EPML namespace. The directory tag contains one EPC model which has the name "Online Shopping" and the

ID “1”. The EPC tag serves as a container of an unordered set of EPC control flow elements. All of the latter have a unique ID attribute. The name tag of the 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 [MN04]. It is also used by Graph eXchange Language (GXL) [WKR02]; by Petri Net Markup Language (PNML) [WK02]; by MS Visio’s XML-based VDX format [Mi03]; XML Metadata Interchange for UML models [OMG03]; and XML Process Definition Language (XPDL) from Workflow Management Coalition (WfMC) [Wf02]. In contrast the Business Process Modeling Language (BPML) [Ar02] and the Business Process Execution Language for Web Services (BPEL4WS) [An03] use a block-oriented representation. AML, the XML format of ARIS Toolset [IDS01] uses adjacency sub-element lists which are attached to the source node of an arc. Arcs and connectors are not required to have a name. A complete list of EPC control flow elements and their sub-elements is presented in Table 2. The following Section illustrates the representation of hierarchical EPC Schemas in EPML.




```
<?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>
            <or id="2"/>
            <arc id="11">
                <flow source="2" target="3"/>
            </arc>
            <function id="3">
                <name>Add Product to
                Shopping Cart</name>
            </function>
            <arc id="12">
                <flow source="3" target="4"/>
            </arc>
            <or id="4"/>
            ...
        </epc>
    </directory>
</epml:epml>
```

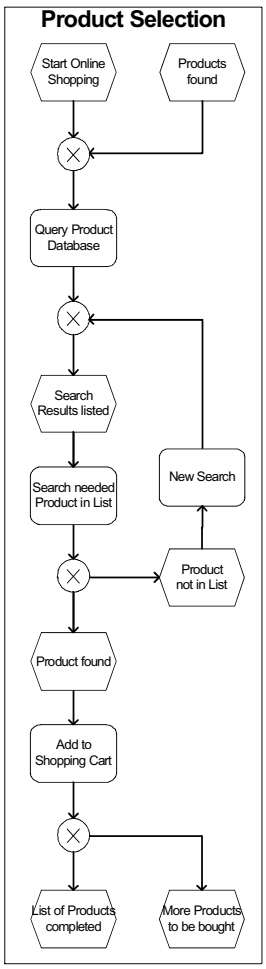
Fig. 3: A simple Online Shopping Process and parts of its EPML representation.

EPML element	Attributes and Sub-Elements
<event>	@id <name> <description> <reference @defRef> ? <graphics> ? <syntaxInfo @implicitType> ?
<function>	@id <name> <description> <reference @defRef> ? <graphics> ? <syntaxInfo @implicitType> ? <toProcess @linkToEpcId> ? <unitReference @unitRef @role> ?
<processInterface>	@id <name> <description> <reference @defRef> ? <graphics> ? <syntaxInfo @implicitType> ? <toProcess @linkToEpcId> ?
<and>, <or>, <xor>	@id <name> ? <description> ? <graphics> ? <syntaxInfo @implicitType> ?
<arc>	@id <name> ? <description> ? <flow @source @target> ? <graphics> ? <syntaxInfo @implicitType> ?

Table 2: Control flow elements of an EPML file.

6. Hierarchical EPCs in EPML

Consider an extension of the example above. After the "Online Shopping" process has been modelled, the function "Add Product to Shopping Cart" is refined by a sub-process called "Product Selection". This means that the EPML file has to include this new process and the hierarchical relation between the function and the sub-process. Figure 4 illustrates the EPML representation of EPC processes with hierarchy relations. The code includes an excerpt from the "Online Shopping" process described in Fig. 3. The hierarchy relation is described via sub-element of the function tag which is called `<toProcess>`. This element has a `@linkToEpcId` pointing to the "Product Selection" process which has an `@epcId` of 2. Hierarchy relations of process interfaces are also described by 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 is unique for the whole EPML file. EPC models may be organized in a hierarchy of directories. Hierarchy relations are allowed between processes no matter 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. I.e. a function used twice in a process hierarchy should contain a `<reference>` sub-element pointing to a function definition that stores its parameters in the definitions block.




```
<?xml version="1.0" encoding="UTF-8"?>
<epml:epml xmlns:epml="http://www.epml.de">
    ...
    <directory name="Root">
        <epc epcId="1"
            name="Online Shopping">
            ...
            <function id="3">
                <name>Add Product to
                Shopping Cart</name>
                <toProcess linkToEpcId="2"/>
            </function>
            <arc id="12">
                <flow source="3" target="4"/>
            </arc>
            <or id="4"/>
            ...
        </epc>
        <epc epcId="2"
            name="Product Selection">
            ...
        </epc>
    </directory>
</epml:epml>
```

Fig. 4: The Product Selection process – a sub-process of the Online Shopping Process.

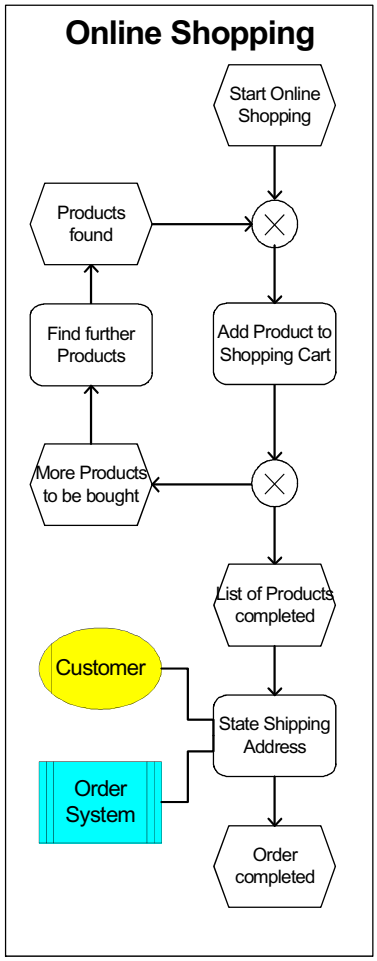
7. Business Perspectives and Views

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 [Fi92]. There have been many different perspectives proposed for business process modelling. The Architecture of Integrated Systems (ARIS) extends the EPC with a data-oriented, a functional, an organisational, an application-oriented, and a product/service-oriented perspective [Sc00]. The PROMET concept differentiates between business dimensions explicitly including organisation, data, functions, and personnel [Ös95]. An in-depth survey of organisational entities provided in workflow management systems is given in [RM98]. The link between role-based access control (RBAC) and business scenarios is analysed in [NS02] and a methodology to generate role hierarchies is developed. From a delegation perspective [AKV03] structure the organisational perspective of a workflow system into a meta model including resources, organisational units, users, and roles. In [Wh03] and [BAN03] 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 [Ch01] of Web Services as a new category of resource perspectives. Beyond resources there have been further perspectives proposed like e.g. risk [BO02], performance measurement [IDS03] to name but a few. The OWL-S Initiative strives to develop a standardised business process ontology for Web Service [OW04]. This is a difficult task taken into consideration the variety of possible perspectives and views. There are even doubts whether a standardised ontology is desirable, because different domains and different business sectors need tailor-made meta models that best fit their specific business model [KK02].

EPML element	Attributes and Sub-Elements
<view>	@name <unit> * <unitRelation> *
<unit>	@unitId @name
<unitRelation>	@relationId @unitRef @subUnitRef @annotation ?
<unitReference>	@unitRef @role ? @value ?

Table 3: Business perspectives and views in EPML.

These arguments have governed the decision of letting EPML be guided by the principle of extensibility instead of standardising certain views. The <view> element is meant to be a container of entities of a certain business perspective and their relationships (cf. Table 3). The <unit> element describes an entity within the domain of a business view by a @unitId and a @name. The <unitRelation> expresses a hierarchical relationship between by the help of a @unitRef and a @subUnitRef. The @annotation may be used to detail the kind of relationship between the units. There is also a @relationId included in order to logically distinguish different relationships between two of the same units. Function elements of a control flow may contain a <unitReference>. The @role and the @value attribute allow one to specify additional information concerning the relationship between the function and the unit.




```
<?xml version="1.0" encoding="UTF-8"?>
<epml:epml xmlns:epml="http://www.epml.de">
    ...
    <view name=”Party>
        <unit unitId=”u1” name=”Customer”/>
    </view>
    <view name=”System”>
        <unit unitId=”u2” name=”Order System”/>
    </view>
    <directory name="Root">
        <epc epcId="1"
            name="Online Shopping">
            ...
            <function id="7">
                <name>State Shipping Address
                </name>
                <unitReference unitRef=”u1”
                    role=”performs activity”/>
                <unitReference unitRef=”u2”
                    role=”receives data record”/>
            </function>
            ...
        </epc>
    </directory>
</epml:epml>
```

Fig. 5: The Online Shopping process including a “Customer” and an “Order System” unit.

Figure 5 illustrates the use of view and unit elements in the “Online Shopping” process of our example. In the header of the EPML file the views “Party” and “System” are declared. Each of these views has one unit: “Customer” is a “Party”, and “Order System” is a “System” involved in the process. Within the function element these units are referenced via a <unitReference> element. The first one describes that the unit “u1” (the customer) takes the role “performs activity” for the function “State Shipping Address”. The second illustrates that the “Order System” receives a data record from this function. This mechanism can be used to declare arbitrary views for an EPC process.

8. Graphical Information

Graphical Information refers to the presentation of EPC models in graphical BPM tools. This is a topic that is not special to EPML. The Petri Net Markup Language (PNML) has worked out and included a proposal for graphical information to be exchanged between modelling tools [Bi03]. 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. Similar to the <graphics> element of control flow objects, the top level element <graphicsDefault> may contain <fill>, <line>, and <font> default settings, but no <position> element.

EPML element	Attributes and Sub-Elements
<graphics>	<position> <fill> <line> <font>
<position>	@x, @y, @width, @height
<fill>	@color, @image, @gradient-color, @gradient-rotation
<line>	@shape, @color, @width, @style
<font>	@family, @style, @weight, @size, @decoration, @color, @verticalAlign, @horizontalAlign, @rotation

Table 4: The graphics element of an EPML file.

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. 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.

The <fill> element describes the appearance of the interior of an object. Arcs do not have fill elements. The @color attribute must take a RGB value or a predefined colour of Cascading Stylesheets 2 (CSS2) [Bo98]. In order to describe a continuous variation of the filling colour an optional @gradient-color may be defined. The @gradient-rotation sets the orientation of the gradient to vertical, horizontal, or diagonal. If there is the URI of an image 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 colour defined. @verticalAlign and @horizontalAlign specify the alignment of the text. In PNML the align attribute corresponds to the EPML horizontalAlign attribute, and verticalAlign is covered by a PNML offset element. @rotation describes a clockwise rotation of the text similar to the concept in PNML.

9. Outlook

Throughout this paper we have outlined how EPML can be used to store an exchange EPC business process models. Yet, there is still much discussion needed within the EPC community to achieve a consensus on EPC representation in EPML, and to leverage EPML application. There are several issues that will be addressed in the future. Firstly, in order to leverage the benefits of EPML as an interchange format, transformation scripts will be developed from major BPM tools towards EPML and reverse. A second issue is the graphical presentation. For PNML there already exists a transformation script to Scalable Vector Graphics (SVG) [Fe03]. A similar script will be developed from EPML to SVG. Thirdly, an XSLT-based [Cl99] syntax checker will be developed and continue the efforts of an XML-based syntax validation of EPCs [MN03c]. Finally, there is still much research needed to come to a general understanding of business perspectives for BPM. Methodologically, this will have to take meta modelling and semantic web techniques into account; furthermore related research on concrete perspectives will have to be consolidated. Administration of decentralized, loosely coupled models will be one of the topics in this context. In this sense, 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. Information, on EPML can be found at http://wi.wu-wien.ac.at/~mendling/EPML/.

References

[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.

[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.

[AKV03] van der Aalst, W.M.P.; Kumar, A.; Verbeek, H.M.W.: Organizational Modeling in UML and XML in the Context of Workflow Systems. In: Proceedings of the 2003 ACM Symposium on Applied Computing (SAC), 2003, pp. 603-608.

[AL02] Arenas, M.; Libkin, L.: A normal form for XML documents. In: Proceedings of the 21st ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'02), 2002, pp. 85-96.

[AL03] Arenas, M.; Libkin, L.: 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), 2003, pp. 15-26.

[AN02] ANSI (ed.): ASC X12 Reference Model for XML Design, July 2002. http://www.x12.org/x12org/comments/X12Reference_Model_For_XML_Design.pdf.

[An03] Andrews, T.; Curbera, F.; Dholakia, H.; Goland, Y.; Klein, J.; Leymann, F.; Liu, K.; Roller, D.; Smith, D.; Thatte, S.; Trickovic, I.; Weerawarana, S.: Business Process Execution Language for Web Services (BPEL4WS) Version 1.1. BEA, IBM, Microsoft, SAP, Siebel, 2003.

[Ar02] Arkin, A.: Business Process Modeling Language (BPML). BPMI.org, 2002.

[BAN03] Becker, J.; Algermissen, L.; Niehaves, B.: Prozessmodellierung in eGovernment-Projekten mit der eEPK. In: Nüttgens, M.; Rump, F.J. (eds.): EPK 2003 - Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten, Proceedings of the GI-Workshop EPK 2003, pp. 31-44.

[Be01] Beech, D.; Lawrence, S.; Moloney, M.; Mendelsohn, N.; Thompson, H.S. (eds.): XML Schema Part 1: Structures. World Wide Web Consortium, Boston, USA, 2001. http://w3c.org/TR/2001/REC-xmlschema-1-20010502/.

[Bi95] Biskup, J.: Achievements of relational database schema design theory revisited. In: Libkin, L.; Thalheim, B.: Semantics in Databases, LNCS 1358, 1998, pp. 29-54.

[Bi03] Billington, J.; Christensen, S.; van Hee, K.E.; Kindler, E.; Kummer, O.; Petrucci, L.; Post, R.; Stehno, C.; Weber, M.: The Petri Net Markup Language: Concepts, Technology, and Tools. In: van der Aalst, W.M.P.; Best, E. (eds.): Applications and Theory of Petri Nets 2003, 24th International Conference, ICATPN 2003, Eindhoven, The Netherlands, June 23-27, 2003, Proceedings. LNCS 2679, 2003, pp. 483-505.

[BM01] Biron, P.V.; Malhotra, A. (eds.): XML Schema Part 2: Datatypes. World Wide Web Consortium, Boston, USA, 2001. http://w3c.org/TR/2001/REC-xmlschema-2-20010502/.

[Bo98] Bos, B.; Lie, H.W.; Lilley, C.; Jacobs, I. (eds.): Cascading Style Sheets, level 2 – CSS2 Specification. http://w3c.org/TR/CSS2, 1998.

[BO02] Brabänder, E.; Ochs, H.: Analyse und Gestaltung prozessorientierter Risikomanagementsysteme mit Ereignisgesteuerten Prozessketten. In: Nüttgens, M.; Rump, F.J. (eds.): EPK 2003 - Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten, Proceedings of the GI-Workshop EPK 2002, pp. 17-34.

[CD99] Clark, J.; DeRose, S.: XML Path Language (XPath) Version 1.0, World Wide Web Consortium, Boston, USA, 1999. http://www.w3.org/TR/1999/REC-xpath-19991116.

[Ch01] Christensen, E.; Curbera, F.; Meredith, G.; Weerawarana, S.: Web Service Description Language (WSDL) 1.1, World Wide Web Consortium, Boston, USA, 2001. http://www.w3.org/TR/wsdl.

[Cl99] Clark, J. (ed.): XSL Transformations (XSLT) Version 1.0. World Wide Web Consortium, Boston, USA, 1999. http://w3c.org/TR/1999/REC-xslt-19991116/.

[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.

[DC03] Dublin Core Metadata Initiative: Dublin Core Metadata Element Set, Version 1.1: Reference Description. 2003. http://dublincore.org/documents/2003/02/04/dces/.

[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.

[De03] Delphi Group (ed.): BPM 2003 – Market Milestone Report. Delphi Group White Paper. Boston, 2003.

[EM01] Embley, D.W.; Mok, W.Y.: Developing XML documents with guaranteed “good” properties. In: Kunii, H.S.; Jajodia, S.; Sølvberg, A. (eds.): Conceptual Modeling - ER 2001, 20th International Conference on Conceptual Modeling, LNCS 2224, 2001, pp. 426 – 441.

[Fe03] Ferraiolo, J.; Jun, F.; Jackson, D. (eds.): Scalable Vector Graphics (SVG) 1.1 Specification. http://www.w3c.org/TR/SVG11, 2003.

[Fi92] Finkelstein, A.; Kramer, J.; Nuseibeh, B.; Finkelstein, L.; Goedicke, M.: Viewpoints: A Framework for Integrating Multiple Perspectives in System Development. In: International Journal of Software Engineering and Knowledge Engineering. 2 (1992) 1, pp. 31-57.

[Ga02] Gartner Research: The BPA Market Catches Another Major Updraft. Gartner's Application Development & Maintenance Research Note M-16-8153, 12 June 2002.

[IDS01] IDS Scheer AG (ed.): XML-Export und-Import mit ARIS 5.0, Stand Januar 2001, Saarbrücken, 2001.

[IDS03] IDS Scheer AG (ed.):ARIS Process Performance Manager, Whitepaper 2003, Saarbrücken. www.ids-scheer.com/sixcms/media.php/ 1186/aris_ppm_whitepaper_e_v500.pdf.

[ISO01] Ketels, K.: ISO 15022 XML Design Rules, Technical Specification, 2001. http://xml.coverpages.org/ISO15022-XMLDesignRulesV23a.pdf.

[Ke99] Keller, G. & Partner: SAP R/3 prozessorientiert anwenden. Iteratives Prozeß-Prototyping mit Ereignisgesteuerten Prozeßketten und Knowledge Maps, Bonn et al. 1999.

[Ki03] Kindler, E.: On the semantics of EPCs: A framework for resolving the vicious circle (Extended Abstract). In: Nüttgens, M.; Rump, F.J. (eds.): EPK 2003 -

Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten, Proceedings of the GI-Workshop EPK 2003, pp. 7-18.

[KK02] Karagiannis, D.; Kühn, H.: 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, September 2002, LNCS 2455, Springer-Verlag, p. 182-196.

[KM94] Keller, G.; Meinhardt, S.: SAP R/3-Analyzer: Optimierung von Geschäftsprozessen auf der Basis des R/3-Referenzmodells, Walldorf, 1994.

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

[Mi03] Microsoft (ed.): About the XML for Visio Schema. MSDN Library, 2003. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/devref/HTML/XMLR_XMLBasics_818.asp

[MN02] Mendling, J.; Nüttgens, M.: Event-Driven-Process-Chain-Markup-Language (EPML): Anforderungen zur Definition eines XML-Schemas für Ereignisgesteuerte Prozessketten (EPK). In: Nüttgens, M.; Rump, F. (eds.): EPK 2002 – Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten, Proceedings of the GI-Workshop EPK 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.: Proc. of the 2nd International Conference on Information Systems Technology and its Applications (ISTA), LNI Vol. P-30, Bonn 2003, 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.

[MN03c] Mendling, J.; Nüttgens, M.: EPC Syntax Validation with XML Schema Languages. In: Nüttgens, M.; Rump, F.J. (eds.): EPK 2003 - Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten, Proceedings of the GI-Workshop EPK 2003, pp. 19-30.

[MN04] Mendling, J.; Nüttgens, M.: XML-based Reference Modelling: Foundations of an EPC Markup Language. In: Becker, J. (ed.): Proceedings of the 8th GI Workshop „Referenzmodellierung“, Essen, Germany.

[NR02] Nüttgens, M.; Rump, J.F.: Syntax und Semantik Ereignisgesteuerter Prozessketten (EPK). In: Desel, J.; Weske, M. (eds.): Promise 2002 - Prozessorientierte Methoden und Werkzeuge für die Entwicklung von Informationssystemen, Proceedings GI-Workshop und Fachgruppentreffen (Potsdam, Oktober 2002), LNI Vol. P-21, Bonn 2002, pp. 64-77.

[NS02] Neumann, G.; Strembeck, M.: A scenario-driven role engineering process for functional RBAC roles. In: 7th ACM Symposium on Access Control Models and Technologies (SACMAT 2002), pp. 33-42.

[OMG03] Object Management Group (ed.): XML Metadata Interchange (XMI) Specification, May 2003, Version 2.0, 2003.

[Ös95] Österle, H.: Business Engineering. Prozess- und Systementwicklung, Band 1, Entwurfstechniken, Berlin, 1995.

[OW04] The OWL Services Coalition (ed.): OWL-S: Semantic Markup for Web Services. Whitepaper Version 1.0. http://www.daml.org/services, 2004.

[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.

[RM98] Rosemann, M.; zur Mühlen, M.: Evaluation of Workflow Management Systems - A Meta Model Approach. In: Australian Journal of Information Systems 6 (1998) 1, pp. 103-116.

[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 modelling, Berlin et al., 2000.

[SW01] SWIFT (ed.): SWIFTStandards XML Design Rules Version 2.3, Technical Specification, 2001. http://xml.coverpages.org/EBTWG-SWIFTStandards-XML200110.pdf.

[Wf02] Workflow Management Coalition (ed.): Workflow Process Definition Interface – XML Process Definition Language, Document Number WFMC-TC-1025, October 25, 2002, Version 1.0, Lighthouse Point, USA, 2002.

[Wh03] White, S.A: Business Process Modeling Notation – Working Draft 1.0, Aug. 25, 2003. BPMI.org, 2003.

[WHB02] Wüstner, E.; Hotzel, T.; Buxmann, P.: 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 2002).

[WK02] M. Weber, E. Kindler: The Petri Net Markup Language. In: H. Ehrig, W. Reisig, G. Rozenberg, and H. Weber (eds.): Petri Net Technology for Communication Based Systems. LNCS 2472, 2002.

[WKR02] Winter, A.; Kullbach, B.; Riediger, V.: An Overview of the GXL Graph Exchange Language. In: Diehl, S. (ed.) Software Visualization - International Seminar Dagstuhl Castle, LNCS 2269, 2001.

ebXML Business Processes - Defined both in UMM and BPSS

Birgit Hofreiter[1]

[1]Department of Computer Science and Business Informatics, University of Vienna
Liebiggasse 4, 1010 Vienna, Austria
E-Mail: birgit.hofreiter@univie.ac.at

Christian Huemer[1,2]

[2] Department of Information Systems, University Duisburg-Essen
Universitätsstr. 9, 45141 Essen, Germany
E-mail: huemer@wi-inf.uni-essen.de

Abstract: The ebXML framework consists of eight specifications for conducting e-Business. The loosely coupled specifications span over the topics of messaging, registries, profiles & agreements, business processes, and core (data) components. The choreography of business processes is defined by instances of the business processes specification schema (BPSS). The BPSS is defined as an XML schema used by business systems to support the execution of business collaborations. It is based on concepts introduced by the UN/CEFACT Modelling Methodology (UMM), or the UMM meta model to be more specific. Thus, BPSS provides the bridge between e-business process modeling and specification of e-business software components. In this paper we show how an ebXML business process is represented in both UMM and BPSS.

1 Introduction

In November 1999 UN/CEFACT and OASIS started the ebXML initiative. The vision of ebXML is to create an electronic marketplace, where businesses can find each other, agree to become trading partners and conduct business. All operations are performed automatically by exchanging XML documents. In order to support the needs of small and medium enterprises (SMEs), ebXML envisions that software industries will deliver commercial off-the-shelf software (COTS) for Business-to-Business (B2B) scenarios to the SMEs. This goal is expressed in the ebXML scenario between a large corporation (Company A) and a SME (Company B) as illustrated in Figure 1. The scenario is described in the ebXML technical architecture specification [UO01].

Company A requests business details from the ebXML registry (step 1) and decides to build its own ebXML-compliant application. Company A submits its own business profile information to the ebXML registry. The business profile submitted to the ebXML registry describes the company's ebXML capabilities and constraints, as well as its supported business scenarios. Company B, which uses an ebXML-compliant shrink-wrapped application, discovers the business scenarios supported by Company A in the registry (step 4). Company B sends a request to Company A stating that they would like to engage in a business scenario (step 5). Before engaging in the scenario, company B submits a proposed business arrangement directly to Company A's ebXML-compliant software interface. The proposed business arrangement outlines the mutually agreed

upon business scenarios and specific agreements. Then, Company A accepts the business agreement. Company A and B are ready to engage in e-business using ebXML (step 6).

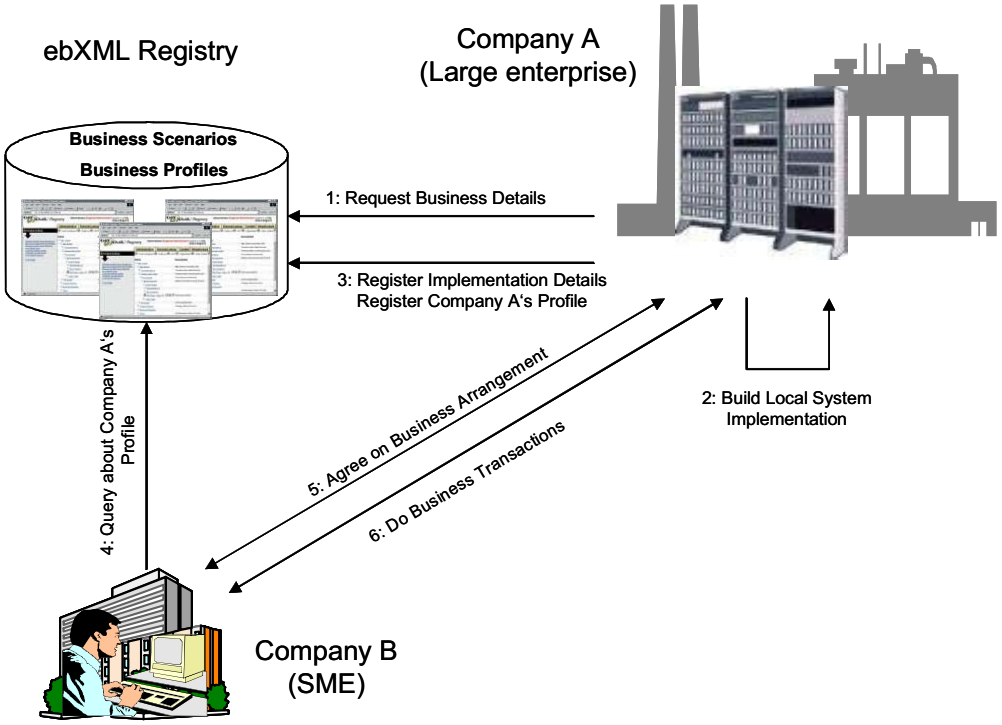


Fig. 1. ebXML Scenario

In order to support this secenario ebXML offers a modular suite of specifications. These specifications provide a standard method to exchange business messages, conduct trading relationships, communicate data in common terms and define/register business processes [HHK02]. The current set of specification comprises the ebXML requirements, technical architecture, messaging service, registry services specification, registry information model, collaboration protocol profile and agreement specification, business process specification schema, and core components.

In this paper we concentrate on the ebXML business process specification schema (BPSS) in its current version 1.10 [UN03c]. It is defined as an XML schema. The BPSS is used to describe standard B2B business processes by defining a choreography of activites amongst business partners. It usally refers to standard business documents that are exchanged in the business process. A partner's profile references the BPSS of a supported process and the role(s) therein. Similarily, a business partners' agreement references the BPSS of a business process in which the business partners will collaborate.

Creating a BPSS does not require a specific process modeling methodology. However, BPSS is based on concepts introduced by UN/CEFACT's Modelling Methodology (UMM). UMM is a methodology for defining the business aspects of a B2B collaboration. It is based on UML [BJR98]. BPSS can be considered as an XML representation of a subset of UMM's meta model. In Section 2 we define an ebXML business process by the means of UMM. The graphical UML syntax helps to quickly

understand the basic concepts. In Section 3 we show the equivalent business process in the XML syntax of BPSS. We show how the UMM concepts are mapped to BPSS and point out the rare cases where BPSS extends the UMM concepts. We conclude with a summary in Section 4.

2 UN/CEFACT's Modelling Methodology

ebXML does not require any specific modeling language or modeling methodology. However, the architecture specification recommends that if implementers and users decide to apply business process modeling, they shall use UMM [UO01]. The UMM concentrates on the business semantics of B2B partnerships. It captures the commitments that are made by business partners when agreeing to a certain type of business processes. These commitments are reflected in the resulting orchestration of the business process involving information exchanges. UMM defines a procedure [UN03b] that describes the necessary steps to create business collaboration models. These business collaboration models are independent of the technology (e.g. ebXML) used to implement them. The term "business collaboration" is used in UMM for a business process involving two or more business partners to accomplish a common business goal.

In addition to the procedure, the reference ontologies, and the patterns, UMM delivers a meta model [UN03a]. The UMM meta model puts the UML meta model into a small corset defining those diagram types that are specific for B2B. The most important diagram types are presented in the subsections below. The UMM procedure [UN03b] leads to business collaboration models that are valid instances of the UMM meta model. These business collaboration models contain more information than what is required for configuring ebXML compliant software, but not less than that. BPSS is based on the UMM meta model. It uses only those concepts that are important for the configuration of the ebXML software. Thus, BPSS is a subset of the UMM meta model, but expressed as XML schema. One might use any methodology to create an BPSS instance. However, UMM guarantees a consistent way for developing a business collaboration model that is a superset of a BPSS instance.

Beforehand, we introduce the UMM by means of a simple example. In the next section the resulting business collaboration model is mapped to the BPSS equivalent. It is not our goal to introduce UMM in all the details. We limit ourselves to those features that help to understand the concepts of BPSS.

The UMM procedure as well as the UMM meta model consists of 4 views in order to describe business collaboration models. Firstly, the Business Domain View (BDV) provides a framework for understanding existing business processes and categorizing these business processes into business areas and process areas. Secondly, the Business Requirements View (BRV) identifies possible business collaborations and further elaborates on these collaborations. It describes processes and resources used to achieve certain objectives and the resulting commitments. In other words, the BRV focuses on the economics of a system. Thirdly, the Business Transaction View (BTV) presents the view of the business process analyst. It defines the orchestration of the business collaboration and structures the business information exchanged. Finally, the Business

Service View (BSV) considers the interaction sequences between network components in order to map the business collaboration semantics to collaborating application systems. The business service view does not add any new information. Its artefacts are automatically created from the information gained in the previous process steps. The BSV artefacts are not mapped to the BPSS. Therefore, we do not detail the BSV in the following subsections.

2.1 Business Domain View (BDV)

The first workflow of UMM is used to gather existing knowledge. It identifies the business processes in the domain of the business problems that are important to stakeholders. It is important at this stage that business processes are not constructed, but discovered. Stakeholders might describe intra-organizational as well as inter-organizational business processes. Both types are recorded. However, the description concentrates on so-called business interface tasks, where an organization communicates with its partners. All the discovered business processes are classified according to a pre-defined classification schema. The final result of the business domain view allows a business process analyst to find opportunities for business collaborations that are constructed in the following view.

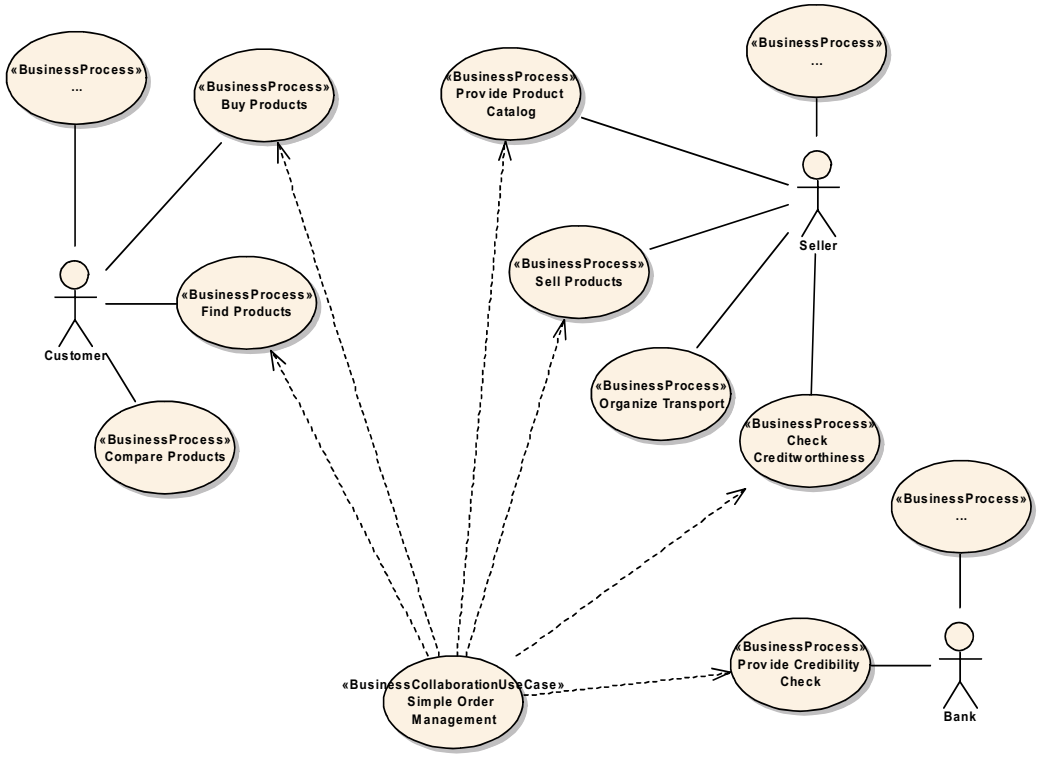


Fig. 2. Business Processes to be Considered in Simple Order Management

To demonstrate UMM and BPSS we use the example of a simple order management process (see also [HH03, HHN04]). Note that this example does not include all the complexity that might be involved in an order management process. The simplified process still allows us to show the main concepts of UMM. In the first workflow the

stakeholders in the simple order management process are interviewed. Customers, sellers, and banks (amongst others not included in the example) describe their business processes that are important in the domain under consideration. These business processes are documented by UML use cases. The resulting use case diagram is depicted in Figure 2. Each actor (customer, seller, and bank) is associated with those business processes that are described by the respective role. The details of each business process are described by completing a worksheet. UMM provides a worksheet type designed to capture all the relevant aspects of a business process. Due to space limitation we do not further concentrate on these worksheets.

2.2 Business Requirements View (BRV)

The goal of the business requirements view is to identify possible business collaborations in the considered domain and to detail the requirements of these collaborations. Business collaborations span over multiple business processes discovered in the previous workflow. Thus, a use case for a business collaboration must consider the views of different stakeholders. The description of the use case must present an harmonized view on the business collaboration being developed.

In our example we identify a *simple order management* as a possible business collaboration between a customer, a seller, and a bank. This *simple order management* depends on business processes described in the previous workflow. The simple order management is described again by a use case together with a worksheet especially designed for business collaboration use cases. We show the *simple order management* use case already at the bottom of Figure 2. This allows us to show the dependency of the business collaboration use case on the business processes of the BDV.

Possible business collaborations identified in the BRV are multiparty as well as binary business collaborations. Binary collaborations are between two business partners only. More than two business partners participate in a multiparty collaboration. Business collaborations might be complex involving a lot of activities between business partners. However, the most basic business collaboration is a binary collaboration realized by a request from one side and an optional response from the other side. This simple collaboration is a unit of work that allows roll back to a defined state before it was initiated. Therefore, this special type of collaboration is called a business transaction.

Use cases and special worksheets document the requirements of multiparty business collaborations, binary business collaborations and business transactions. So-called business collaboration protocol use cases define the needs of both multiparty and binary business collaborations. The term “protocol” appears in the stereotype to express that a complex protocol is needed to choreograph the activities of the collaboration. Less surprisingly, a business transaction use case describes the requirements of a business transaction. A business collaboration protocol use case usually requires other business collaborations and/or business transactions to be included as part of it. This fact is denoted by “include”-relationships between the respective use cases (c.f. Figure 4). In the BRV all the business collaboration use cases are decomposed recursively until the lowest level, which is a business transaction use case, is reached.

The *simple order management* collaboration is a multiparty collaboration since three business partners (customer, seller, bank) are participating in this collaboration. BPSS allows multiparty collaborations only if they are synthesized from two or more binary collaborations. A multiparty collaboration is not able to reflect dependencies between intermediate states of different business collaborations. For a better understanding we anticipate what we will see later on: The purchasing process between a customer and a seller involves multiple choreographed steps. Somewhere in the course of this process a customer requests registration. The seller might check the credibility of the customer with the customer's bank before responding to the customer. This means that some activities of one binary collaboration are performed during the course of the other. By strictly separating the two binary collaborations we loose this choreography information.

UMM is a little bit less restrictive than BPSS. A choreography for a collaboration might include more than two parties. Important in UMM is that a business collaboration is finally decomposed into business transactions, which are by default binary. These business transactions do not overlap. This means that UMM does not support nested business transactions. Note, that the UMM reference guide (currently only available as Chapter 8 of the old UMM Revision 10) mentions that business transactions can be nested - however the UMM meta model does not support it. Coming back to the example above, the credibility check is nested in the registration transaction. This cannot be expressed according to the UMM meta model. Nesting business transactions usually appear in multiparty collaborations. Only, binary collaborations are choreographed without the need for nesting transactions. In our example, we split the multiparty collaboration simple order management into two binary collaborations: (1) the *simple purchase management* performed by customer and seller and (2) the *credibility check* performed by seller and bank. Figure 3 shows this decomposition.

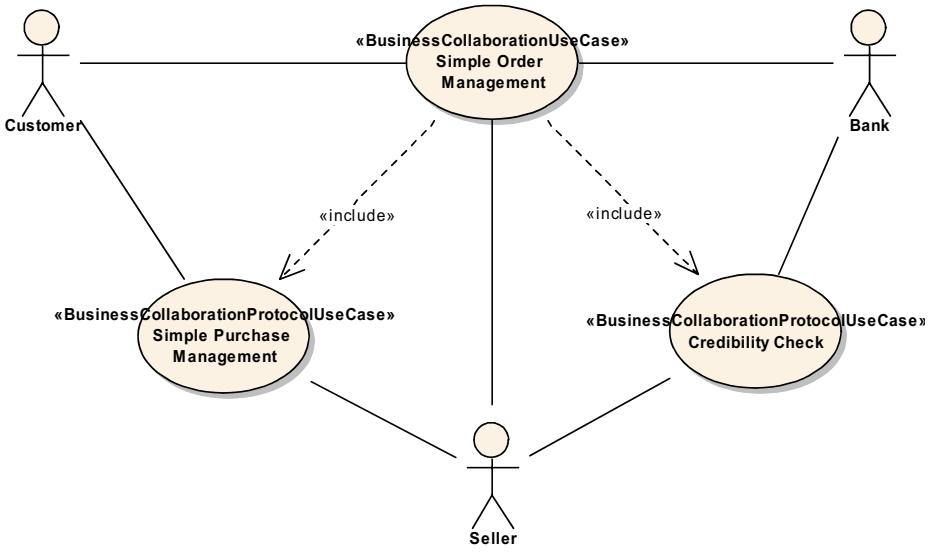


Fig. 3. Decomposing Multiparty Collaborations into Binary Collaborations

On first glance one might consider the weak support of multiparty collaborations as a weak point of UMM. It is necessary to understand that UMM is especially designed for modelling B2B partnerships focusing on the aspects regarding the making of business

decisions and commitments among the business partners. Therefore, a UMM-compliant business collaboration model designs all the steps necessary to fulfill a contract. The business collaboration considers all the parties fulfilling the contract. Most of the contracts in real business life are between two parties. Returning to the example above, a contract is made between the customer and the seller. This contract is independent of whether the seller decides to check the credibility or not. UMM does not standardize any internal processes or decisions.

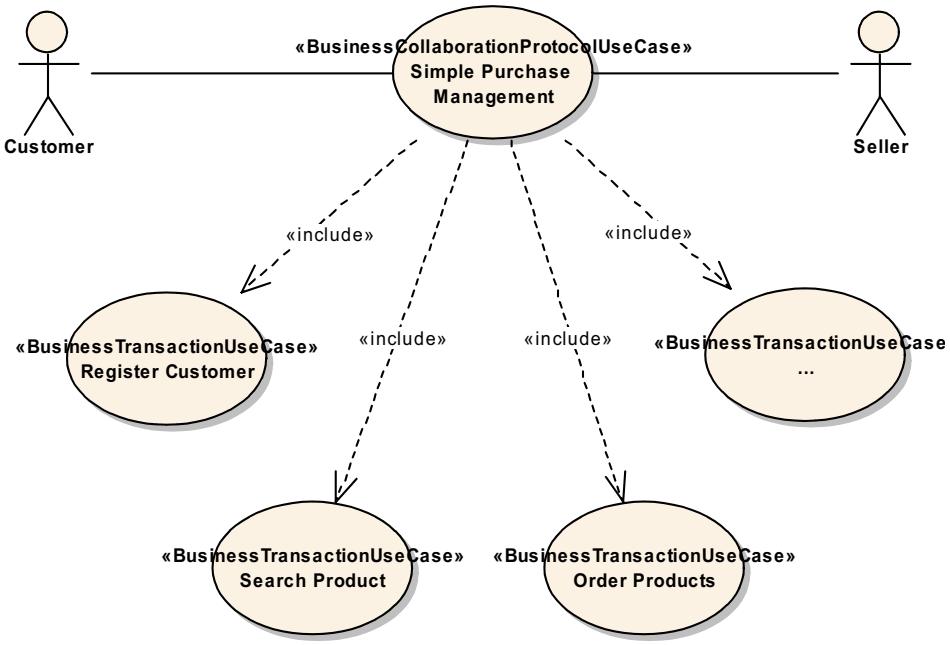


Fig. 4. Decomposing a Binary Collaboration into Business Transactions

The binary collaboration *simple purchase management* includes - according to the work-sheet - the use cases of *register customer*, *search product*, and *order products* among others not detailed here. Figure 4 shows the decomposition. The requirements of these use cases are documented by corresponding worksheets. By completing the worksheet it is easy to detect that each of these use cases can be realized by a business transaction. Consequently, the use cases are stereotyped as business transaction use cases. It follows, that the bottom layer is reached and no more decomposition is necessary.

According to UMM, a business collaboration is best designed by a choreography of business state changes. Thus, it is important to analyze the effects of activities on the business state of the collaboration or, better, on the business states of business entities, which have a life-cycle during the business collaboration. Figure 5 depicts the business state changes in the business transaction *register customer*. It defines preconditions, post-conditions and inter-mediate states of the *customer information* during the transaction.

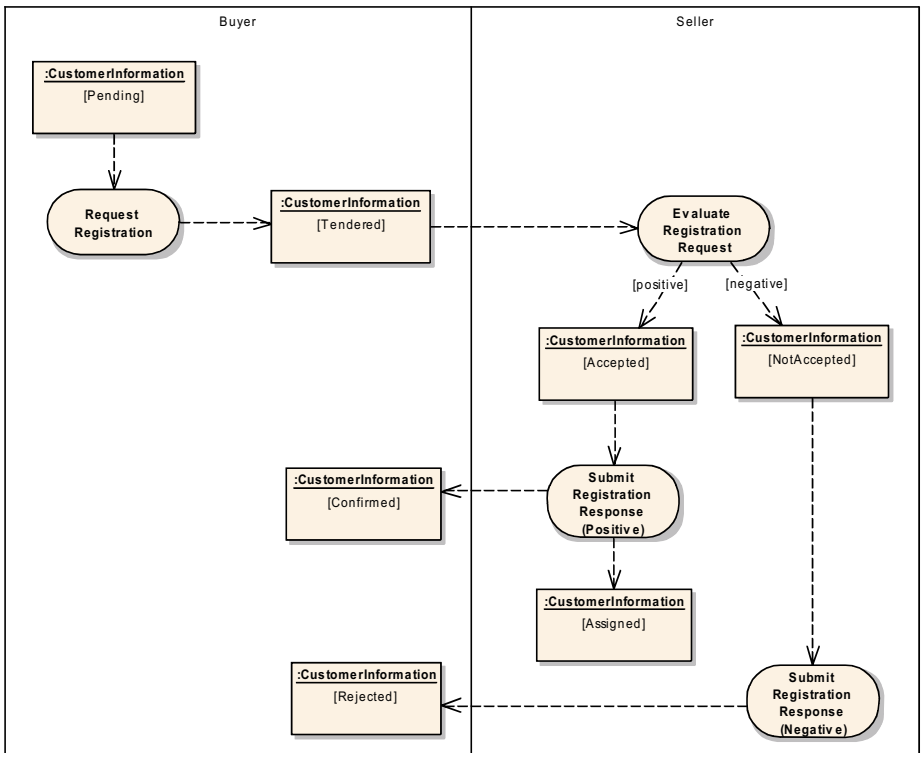


Fig. 5. Business Entity States for Customer Information

2.3 Business Transaction View (BTV)

The BTV defines the choreography for the business collaboration as well as the structure of the business information exchanges. The business transaction view covers three main types of artefacts: the business collaboration protocol, the business transaction diagram and the class diagram on business information structures.

The first step of the BTV models the business collaboration according to the corresponding use case description. The resulting activity graph is called business collaboration protocol. An extract of the business collaboration protocol for the binary collaboration *simple purchase management* is presented in Figure 6. The use case diagram of the BRV in Figure 4 defines that simple purchase management includes *register customer*, *search products* and *order products* among others. All the included use cases are mapped in the BTV to activities of the business collaboration protocol. Business transaction use cases are mapped to business transaction activities and business collaboration protocol use cases to collaboration activities. Each business transaction activity is further detailed by the activity graph for a business transaction (see further below). A collaboration activity is refined by another business collaboration protocol. However, UMM mentions the concept of collaboration activity, but does not support it by the current meta model. It follows that a collaboration activity must be recursively replaced by its detailing business collaboration protocol. Finally, a business collaboration protocol is built by business transactions activities only. We think that the concept of collaboration activities - which is supported by BPSS - is certainly useful and should be considered in future revisions of the UMM meta model.

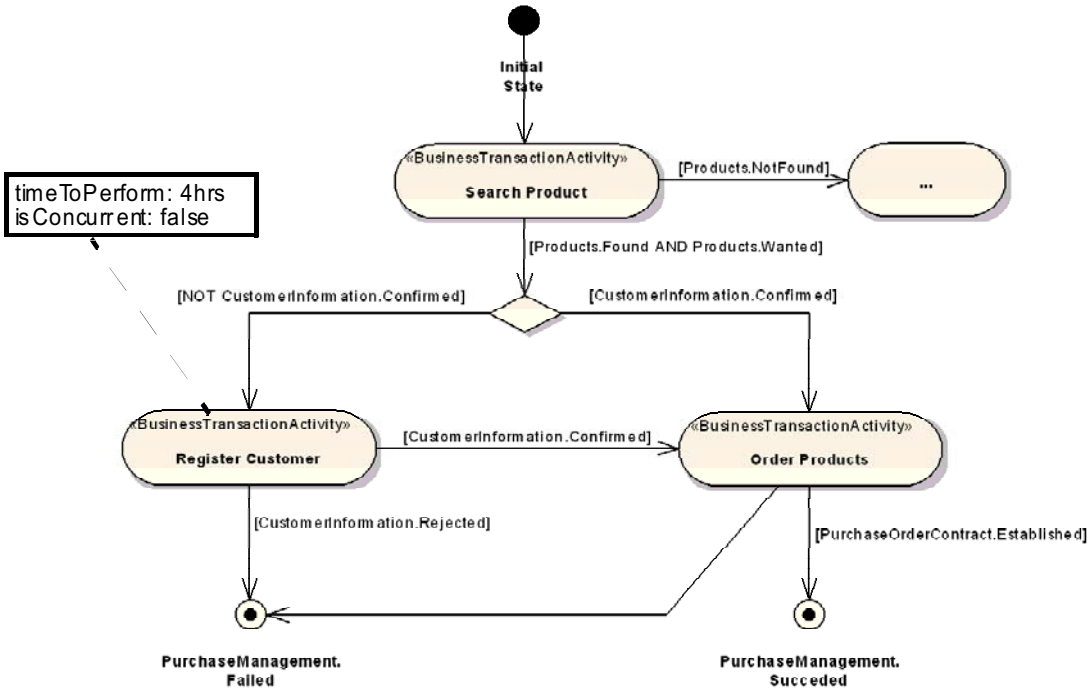


Fig. 6. Business Collaboration Protocol for Simple Purchase Management

For each business transaction activity the maximum performance time is documented by the *timeToPerform* property. If this time is exceeded, the initiating partner has to send a failure notice. Furthermore, the *isConcurrent* property defines whether or not more than one business transaction activity can be open at one time. In our example *register customer* is not concurrent and most be performed in 4 hours at most.

The business collaboration protocol defines the choreography amongst the business transaction activities. The transitions between the activities are guarded by business states of business entities. For example, a transition from *search product* to *order product* requires that products were found and that these products are wanted (by the customer). Furthermore, the customer must be registered at the seller, i.e. she must have a confirmed customer information.

The next step in the BTV is to detail each business transaction activity by its own activity graph. This graph defines the choreography of a business transaction. The activity graph of a business transaction is always composed of two business actions, a requesting business activity performed by the initiator and a responding business activity performed by the other business partner. We stick here to the UMM terms, although initiating and reacting business activities would be better terms, since not each transaction is a request / response one. In the UML notation, a business action is assigned to the swimlane of the respective business partner. In a one-way transaction business information is exchanged only from the requesting business activity to the responding business activity. In case of two-way transaction the responding business activity returns business information to the requesting business activity. The exchange of business information is shown by object flows. Figure 7 shows the business transaction *register customer*.

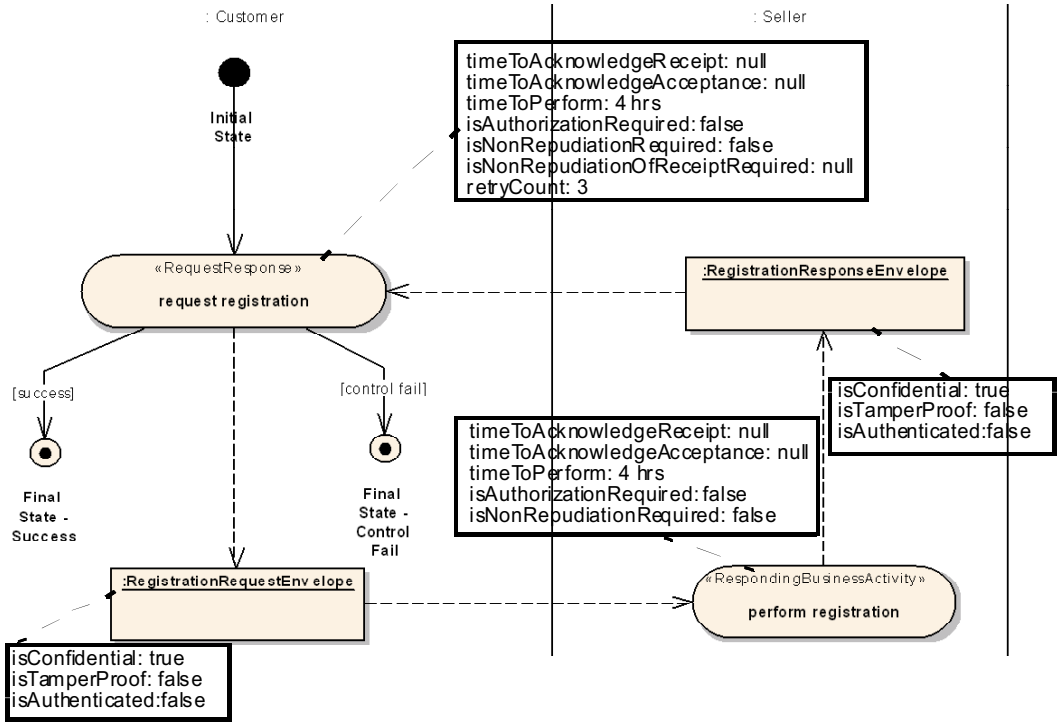


Fig. 7. Business Transaction for Register Customer

In UMM we distinguish two types of one-way transactions. If the business information sent is a formal non-reputable notification, the transaction is called notification. Otherwise the transaction is known as information distribution. Furthermore, there exist four different types of two-way transactions. If the responder already has the information available, it is a query/response transaction. If the responder does not have the information, but no pre-editor context validation is required before processing, the transaction is a request/confirm one. If the latter is required, the next question is whether the transaction results in a residual obligation between the business partners to fulfill terms of a contract. In case of a "yes" it is a commercial transaction and a request/response transaction otherwise. These types of business transactions cover all known legally binding interactions between two decision making applications as defined in Open-edi [ISO95]. They have proven to be useful in RosettaNet. In the UML notation the requesting business activity is stereotyped according to the transaction type.

The different types of business transaction patterns differ in the default values for the parameters that characterize the activities in the business transaction: *timeToAcknowledgeReceipt*, *timeToAcknowledgeAcceptance*, *timeToPerform*, *isAuthorizationRequired*, and *isNonRepudiationRequired.* The values for *isNonRepudiationOfReceiptRequired* and for *retryCount* are only defined for the requesting business activity. Note, an acknowledge of receipt is sent after grammar validation, sequence validation, and schema validation. An acknowledge of acceptance is sent after an additional content validation. Retry count is the number of retries in case of control failures.

The object flow states of a business transaction refer to instances of business information envelopes. The business information envelope and the included business information is modeled in a class diagram. The business information is commonly called a business document. Since UMM is not a document-centric approach and only exchanges information necessary for a business state change, we stick to the term business information.

In order to guarantee reusability, the business information must be built by common building blocks. Unfortunately, the current version of UMM does not reflect this requirement. The meta model only defines that the business information exchanged is built by recursively structured information entities. Recently it was agreed that the business information should be based on ebXML core components. A core components is defined as "a building block that contains pieces of business information that belong to a single concept. Core components are characterized by the fact that they appear in many different circumstances of business information and in many different areas of business." [UN03d] Currently, UN/CEFACT is building a library of core components which will become the richest and cross-industry harmonized source for assembling business information.

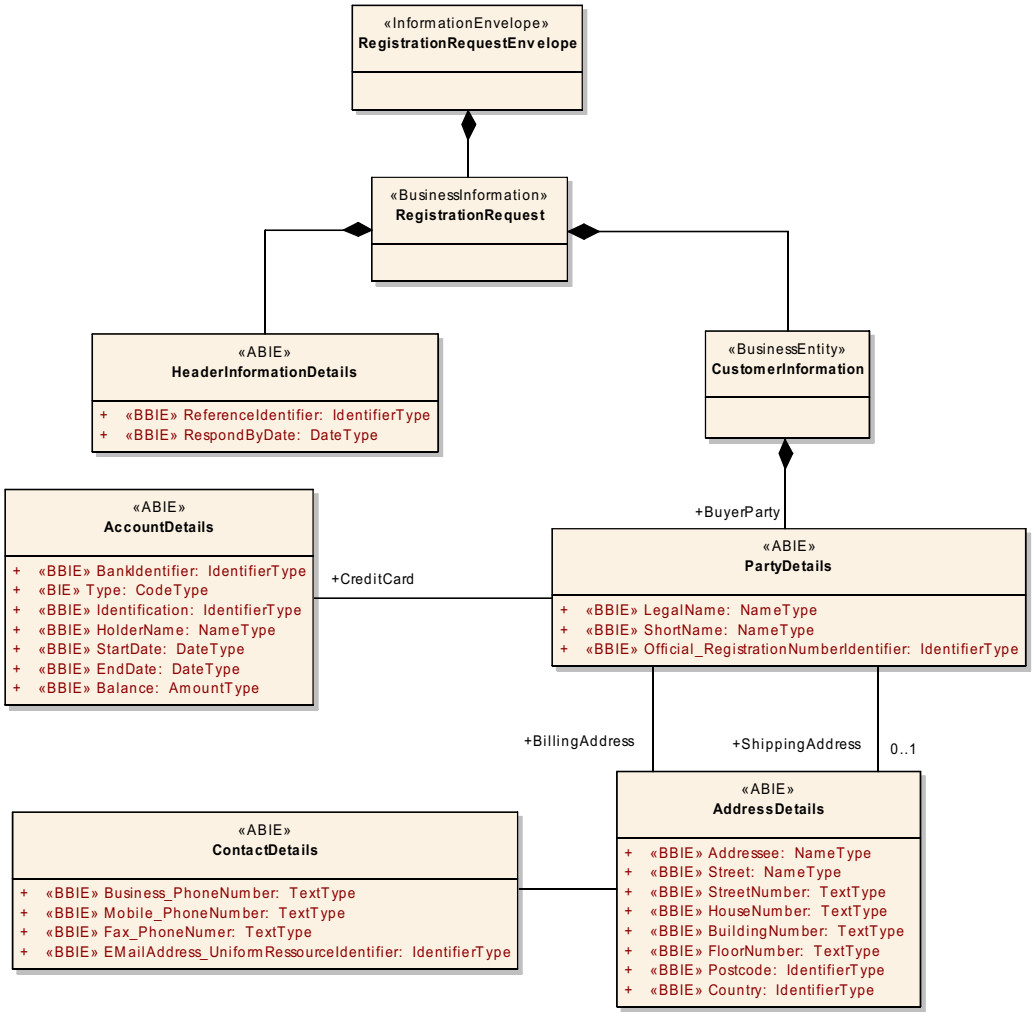


Fig. 8. Class Diagramm for Registration Request

In order to model the class diagram for a business information exchange, one must determine the business entities that are effected by the transaction. Each business entity is described by the information needed to change its business state. This information is built by re-using core components of the library. Thus, one must select suitable core components from the library and customize them to the needs of the business transaction. Customizing means setting the core components into the context of the business transaction. This is when the core component becomes a so called business information entity. In the resulting class diagram a basic business information entity (BBIE) is represented by an attribute and an aggregate business information entity by a class or a group of classes.

Figure 8 presents a class diagram for the registration request. The registration request envelope includes the business information, i.e. the registration request. The registration request aggregates the header details and the affected business entities. In this case the only business entity is the customer information. The customer information must include all the information necessary to change its state from "tendered" to "accepted" and finally "confirmed". Therefore existing core component types are set into the context of our registration transaction. Setting into context means e.g. selecting only those attributes from the more than 20 attributes that are assigned to the core component address. Due to space limitation we do not explain this process in detail but refer to the core components specification [UN03d].

3 From UMM to BPSS

In this section we describe how the UMM model developed in the previous section is equally represented in a BPSS instance. As mentioned earlier BPSS is based on a subset of the UMM meta model. This subset is more or less identical to the Business Transaction View (BTV). Thus, BPSS might be viewed as an "XML-ification" of the BTV artefacts: business collaboration protocol and business transaction with references to exchanged documents. The UMM is an approach intended to specify business collaborations from top down, re-using existing lower level content as much as possible. In BPSS it does not matter whether an top-down or bottom-up approach is used. For educational purposes we use a bottom up approach to describe the concepts of BPSS.

3.1 Business Documents / Business Information

In BPSS, the business information exchanged in a business transaction is called a business document. Thus the UMM concept of business information must be mapped to an BPSS equivalent. BPSS does not by itself support the definition of business document types, nor does any other ebXML specification. It is assumed that the various business document standard organizations will define the rules on how to map the business information entities to their specific document types.

A BPSS instance points to the resulting business document types. If the document type is XML-based, BPSS will use the business document element to point to an external schema. We assume that this is the case in our example below. However, it is possible to define documents of any other structure, e.g. UN/EDIFACT, or completely unstructured

documents as attachments. The following two code fragments show the XML schema for the attribute group *name* used in *BusinessDocument* and many other BPSS elements, and the XML schema for *BusinessDocument*. Each defined business document carries a globally unique Id (GUID) in the *nameID* attribute. The logical name of the business document, which corresponds to the class name for business information in UMM, is defined in the *name* attribute. Either *specificationLocation* or *specificationID* are used to point to the external schema.

```
<xsd:attributeGroup name="name">
    <xsd:attribute name="name" type="xsd:string" use="optional"/>
    <xsd:attribute name="nameID" use="required">
        <xsd:simpleType> <xsd:restriction base="GUID"/> </xsd:simpleType>
    </xsd:attribute>
</xsd:attributeGroup>

<xsd:element name="BusinessDocument">
    <xsd:complexType>
        <snip/>
        <xsd:attributeGroup ref="name"/>
        <xsd:attribute name="specificationLocation" type="xsd:anyURI"/>
        <xsd:attribute name="specificationID" type="xsd:anyURI"/>
        <snip/>
    </xsd:complexType>
</xsd:element>
```

The valid instances for pointing to external XML schemas that define the document types for registration request and registration response are the following:

```
<BusinessDocument nameID="BPSS-XML4BPM-Document-001" name="RegistrationRequest"
specificationLocation="http://www.xml4bpm.org/RegistrationRequest.xsd"/>
<BusinessDocument nameID="BPSS-XML4BPM-Document-002" name="RegistrationResponse"
specificationLocation="http://www.xml4bpm.org/RegistrationResponse.xsd"/>
```

3.2 Business Transactions

BPSS uses the concept of business transactions more or less identically to UMM. This means a business transaction is built by two business actions - the requesting business activity and the responding business activity. The request document flow is mandatory and the responding one is optional. We recommend to read this subsection in conjunction with Figure 7 showing a UMM business transaction.

The information exchanged in a UMM business transaction is represented by an object flow state. This object flow state is an instance of a business information envelope which covers the business information. For this purpose, BPSS defines the element *Document-Envelope*. The envelope has a GUID (attribute *nameID*) and a logical name (attribute *name*). Its attribute *businessDocumentIDREF* references one of the business documents described above. Furthermore, the attribute *businessDocument* contains the logical name of this document. In the example further below the business document envelope *RegistrationRequestEnvelope* is created with its unique Id (*BPSS-XML4BPM-Envelope-000001*). This envelope references the business document *RegistrationRequest* (*BPSS-XML4BPM-Document-000001*).

Both BPSS and UMM define the security parameters *isAuthenticated*, *isConfidential*, and *isTamperDetectabel* (in UMM: *isTamperProof*) for the information exchanged. In UMM these parameters are booleans. BPSS uses a more sophisticated differentiation with four possible values: *none*, *transient*, *persistent*, and *transient-and-persistent*. Transient security focuses on the delivery to the receiving message service handler. Persistence security applies as soon as the document leaves the receiving message handler. Transient security is what is considered by UMM. Therefore, a UMM *true* for a security parameter maps to *transient* in BPSS. Persistence security cannot be mapped automatically. In UMM, each subpart within a business information envelope might have its own security parameters. In BPSS the security parameters are the same for the envelope and its content. By mapping UMM to BPSS the highest security identified for a subpart must be set for the whole envelope. In our UMM example we already used the security parameters only on the envelope level. Therefore, the code fragment for the *registration request envelope* looks as follows (c.f. Figure 7): *isConfidential* is set to *transient*, whereas *isAuthenticated* and *isTamperDetectable* are set to *none*.

```
<xsd:element name="DocumentEnvelope">
    <xsd:complexType>
        <snip/>
        <xsd:attributeGroup ref="name"/>
        <xsd:attribute name="businessDocument" type="xsd:string" use="required"/>
        <xsd:attribute name="businessDocumentIDREF" type="GUIDREF"/>
        <xsd:attribute name="isPositiveResponse" type="xsd:boolean"/>
        <xsd:attributeGroup ref="documentSecurity"/>
    </xsd:complexType>
    <snip/>
</xsd:element>

<xsd:attributeGroup name="documentSecurity">
    <xsd:attribute name="isAuthenticated">
        <xsd:simpleType>
            <xsd:restriction base="xsd:NMTOKEN">
                <xsd:enumeration value="none"/> <xsd:enumeration value="transient"/>
                <xsd:enumeration value="persistent"/><xsd:enumeration value="transient-and-persistent"/>
            </xsd:restriction>
        </xsd:simpleType>
    </xsd:attribute>
    <xsd:attribute name="isConfidential">
        <xsd:simpleType> <snip>identical to isAuthenticated</snip></xsd:simpleType>
    </xsd:attribute>
    <xsd:attribute name="isTamperDetectable">
        <xsd:simpleType> <snip>identical to isAuthenticated</snip></xsd:simpleType>
    </xsd:attribute>
</xsd:attributeGroup>

<DocumentEnvelope nameID="BPSS-XML4BPM-Envelope-001" name="RegistrationRequestEnvelope"
businessDocument="RegistrationRequest" businessDocumentIDREF="BPSS-XML4BPM-Document-001"
isAuthenticated="none" isConfidential="transient" isTamperDetectable="none"/>
```

Next, we consider the requesting activity and the responding activity. Both are subtypes of business action in UMM and BPSS. Each business action has a GUID and a name. In BPSS, the requesting business activity includes the child element document envelope created by the requesting business activity itself. Since the requesting flow is mandatory, the child element is mandatory as well. The responding activity includes an optional document envelope. Note, there is a bug in the BPSS 1.10, since a responding business

activity might output even more document envelopes. We corrected this in the code fragment below. In BPSS there is no explicit link between a business document envelop and the business action that receives this input. The receiving business action is always the one that does not create the business document envelope.

The most significant difference to UMM is the fact, that BPSS does not assign any roles to the business actions on the transaction level. In addition, the parameters characterizing the business actions differ in UMM and BPSS. *timeToAcknowledgeReceipt* and *timeToAcknowledgeAcceptance* are used similar. A value is assigned to each of these attributes, if the acks are required. In UMM an acknowledgment of receipt is sent after schema validation. The BPSS equivalent requires the flag *isIntelligibleCheck* to be set, otherwise the ack is sent after receiving the information without any validation. In BPSS there is no attribute like *timeToPerform*, since the only significant time is the one for the overall transaction that is defined in the business transaction activity. Both *isNonRepudiationRequired* and *isNonRepudiationOfReceiptRequired* are attributes of both business action types - in UMM the latter does not exist for the responding business activity. The *retryCount* for the requesting business activity is also used as in UMM. The *isAuthorizationRequired* attribute is defined, but deprecated, because it cannot be supported by current ebXML business service interfaces.

In our example (c.f. Figure 7) the requesting business activity is *request registration*. Both non-repudiation requirements do not apply. The retry count is set to 3. Time values for the acknowledgments must not be specified, since no acks are sent. *Request registration* outputs the *registration request envelope*. The responding business activity is *perform registration*. Non-repudiation is not required. The acks attributes must be omitted as well. Since both acknowledgments do not require acknowledgments of receipt, the *isIntelligibleCheckRequired* parameter is not useful.

```
<xsd:complexType name="BusinessAction">
    <snip/>
    <xsd:attributeGroup ref="name"/>
    <xsd:attribute name="isAuthorizationRequired" type="xsd:boolean" default="false">
        <xsd:annotation> <xsd:documentation>deprecated</xsd:documentation> </xsd:annotation>
    </xsd:attribute>
    <xsd:attribute name="isIntelligibleCheckRequired" type="xsd:boolean" default="false"/>
    <xsd:attribute name="isNonRepudiationRequired" type="xsd:boolean" default="false"/>
    <xsd:attribute name="isNonRepudiationReceiptRequired" type="xsd:boolean" default="false"/>
    <xsd:attribute name="timeToAcknowledgeReceipt" type="xsd:duration"/>
    <xsd:attribute name="timeToAcknowledgeAcceptance" type="xsd:duration"/>
    </xsd:complexType>
<xsd:element name="RequestingBusinessActivity">
    <xsd:complexType>
        <xsd:complexContent>
            <xsd:extension base="BusinessAction">
                <xsd:sequence><xsd:element ref="DocumentEnvelope"/></xsd:sequence>
                <xsd:attribute name="retryCount" type="xsd:int"/>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
</xsd:element>
```

```
<xsd:element name="RespondingBusinessActivity">
    <xsd:complexType>
        <xsd:complexContent>
            <xsd:extension base="BusinessAction">
                <xsd:sequence>
                    <xsd:element ref="DocumentEnvelope" minOccurs="0" maxOccurs="1"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
</xsd:element>

<RequestingBusinessActivity nameID="BPSS-XML4BPM-Action-001" name="RequestRegistration"
retryCount="3" isNonRepudiationRequired="false" isNonRepudiationReceiptRequired="false">
    <DocumentEnvelope nameID="BPSS-XML4BPM-Envelope-001" name="RegistrationRequestEnvelope" ... />
</RequestingBusinessActivity>
<RespondingBusinessActivity nameID="BPSS-XML4BPM-Action-002" name="PerformRegistration"
isNonRepudiationRequired="false">
    <DocumentEnvelope nameID="BPSS-XML4BPM-Envelope-002" name="RegistrationResponseEnvelope" .../>
</RespondingBusinessActivity>
```

A business transaction is defined by the sequence of the requesting business activity and the responding business activity. In UMM each business transaction follows one out of six legally binding business patterns. The attribute *pattern* of the BPSS element *BusinessTransaction* specifies the underlying pattern type. Furthermore, the attribute *isGuaranteedDeliveryRequired* signals that reading partners must employ a delivery channel that provides a delivery guarantee. UMM always assumes guaranteed delivery if necessary.

The business transaction *register customer* (c.f. Figure 7) includes the requesting business activity *request registration* and the responding business activity *perform registration*. Its underlying pattern is a *query/response transaction*. Furthermore, we assume a guaranteed delivery channel.

```
<xsd:element name="BusinessTransaction">
    <xsd:complexType>
        <xsd:sequence>
            <snip/>
            <xsd:element ref="RequestingBusinessActivity"/>
            <xsd:element ref="RespondingBusinessActivity"/>
        </xsd:sequence>
        <xsd:attributeGroup ref="name"/>
        <xsd:attribute name="pattern" type="xsd:anyURI"/>
        <xsd:attribute name="isGuaranteedDeliveryRequired" type="xsd:boolean" default="false"/>
    </xsd:complexType>
    <snip/>
</xsd:element>

<BusinessTransaction nameID="BPSS-XML4BPM-Transaction-001" name="RegisterCustomer"
pattern="QueryResponseActivity" isGuaranteedDeliveryRequired="true">
    <RequestingBusinessActivity nameID="BPSS-XML4BPM-Action-001" name="RequestRegistration" ... >
        <snip/>
    </RequestingBusinessActivity>
    <RespondingBusinessActivity nameID="BPSS-XML4BPM-Action-002" name="PerformRegistration" ... >
        <snip/>
    </RespondingBusinessActivity>
</BusinessTransaction>
```

```
<BusinessTransactionActivity name="RegisterCustomer" nameID="BPSS-XML4BPM-Activity-001"
businessTransaction="RegisterCustomer" businessTransactionIDREF="BPSS-XML4BPM-Transaction-001"
fromRole="customer" fromRoleIDREF="BPSS-XML4BPM-Role-001" toRole="seller" toRoleIDREF="BPSS-
XML4BPM-Role-002" isConcurrent="false" timeToPerform="PT4H" preCondition="some text" beginsWhen="some
text" endsWhen="some text" postCondition="some text"/>
```

3.3 Binary Collaboration

A binary collaboration is always conducted by two roles. They perform one or more business activities. In BPSS, a business activity is either a business transaction activity or a collaboration activity. We first concentrate on business transaction activities. Collaboration activities are briefly explained at the end of this subchapter. A business transaction activity points to a business transaction. The semantic of a business transaction activity is the same as in UMM (c.f. Figure 6). The business transaction activity is identified by its own *nameID* and *name*. It is quite common that the name of the referenced business transaction is identical. However, this is not a must. The IDREF to the business transaction ensures unambiguouty. For each business transaction activity an initiating role (*fromRole*) and an reacting role (*toRole*) together with IDREFs to the role definitions are specified. The *fromRole* will perform the requesting business activity of the business transaction. The responding business activity is performed by the *toRole*. The UMM parameters *isConcurrent* and *timeToPerform* characterizing a business transaction activity exist in BPSS as well. Further attributes assigned to the BPSS business transaction activity are *preCondition*, *beginsWhen*, *endsWhen* and *postCondition*. Values for these attributes are usually defined in the transaction use case template (worksheet) of UMM.

```
<xsd:complexType name="BusinessActivity">
    <xsd:attributeGroup ref="name"/>
    <xsd:attribute name="fromRole" type="xsd:string" use="required"/>
    <xsd:attribute name="fromRoleIDREF" type="GUIDREF"/>
    <xsd:attribute name="toRole" type="xsd:string" use="required"/>
    <xsd:attribute name="toRoleIDREF" type="GUIDREF"/>
    <xsd:attribute name="beginsWhen" type="xsd:string"/>
    <xsd:attribute name="endsWhen" type="xsd:string"/>
    <xsd:attribute name="preCondition" type="xsd:string"/>
    <xsd:attribute name="postCondition" type="xsd:string"/>
</xsd:complexType>

<xsd:element name="BusinessTransactionActivity">
        <xsd:complexType>
        <xsd:complexContent>
            <xsd:extension base="BusinessActivity">
                <snip/>
                <xsd:attribute name="businessTransaction" type="xsd:string" use="required"/>
                <xsd:attribute name="businessTransactionIDREF" type="GUIDREF"/>
                <xsd:attribute name="isConcurrent" type="xsd:boolean" default="true"/>
                <snip/>
                <xsd:attribute name="timeToPerform" type="xsd:duration"/>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
</xsd:element>
```

The code fragment further below shows the definition of the register customer business transaction activity (c.f. Figure 6). It is a non-concurrent business transaction activity which must be executed within 4 hours. The customer initiates the referenced business transaction, which is defined in the previous subsection. The seller is the responder.

```
<BusinessTransactionActivity name="RegisterCustomer" nameID="BPSS-XML4BPM-Activity-001"
businessTransaction="RegisterCustomer" businessTransactionIDREF="BPSS-XML4BPM-Transaction-001"
fromRole="customer" fromRoleIDREF="BPSS-XML4BPM-Role-001" toRole="seller" toRoleIDREF="BPSS-
XML4BPM-Role-002" isConcurrent="false" timeToPerform="PT4H" preCondition="some text" beginsWhen="some
text" endsWhen="some text" postCondition="some text"/>
```

In UMM the business collaboration protocol choreographs the business transaction activities. This choreography is defined in BPSS by a binary collaboration. Figure 9 shows the subelement structure for a binary collaboration. Due to space limitation we do not present the XML schema code for each of these elements. The details of the binary collaboration are best explained by means of the example depicted in the business collaboration protocol of Figure 6 and mapped to the code fragment further below.

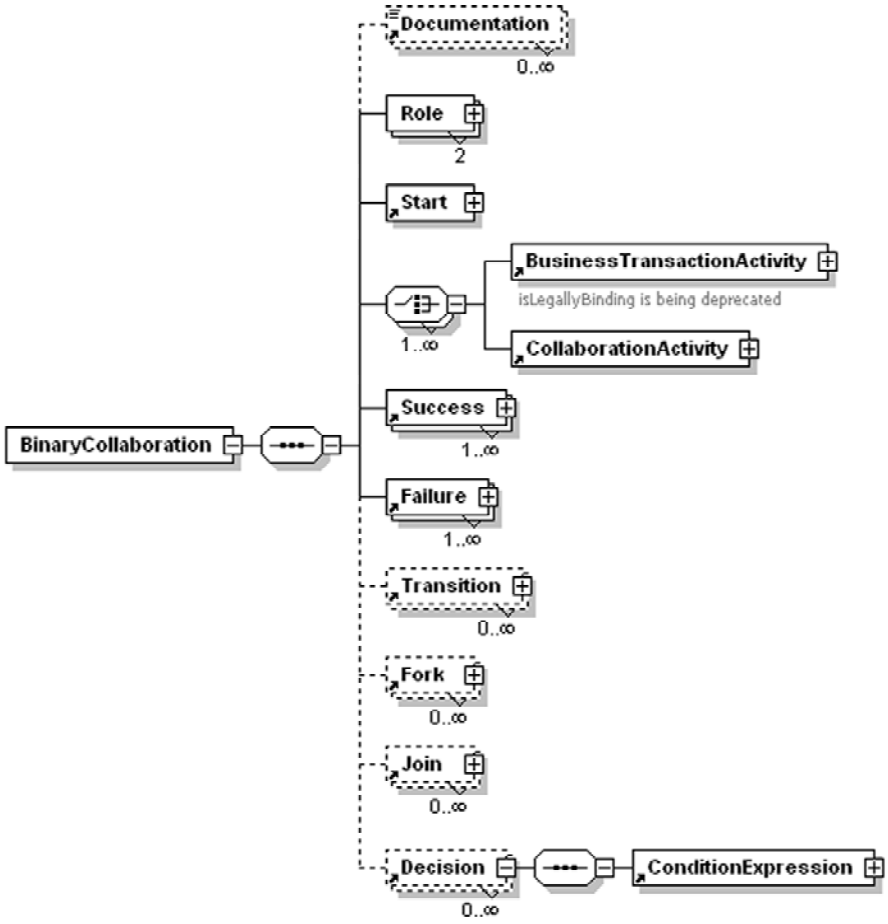


Fig. 9. XML Schema for a Binary Collaboration

The binary collaboration simple purchase management is identified by a unique id (BPSS-XML4BPM-BinaryCollaboration-001). The attribute *isInnerCollaboration* is a flag indicating that the binary collaboration can only be used as subpart of another binary

collaboration. Since this is not the case in our example it is set to false. The *initiating-RoleIDREF* references the role that initiates the first business transaction activity. This attribute is only relevant if a binary collaboration is subpart of another one (see further below). Another attribute indicates a pattern of a binary collaboration. Since these types of patterns are currently in development, this attribute is for future use only. A maximum time for a binary collaboration is defined by *timeToPerform* - a concept that does not exist on this level in UMM. The attributes *preCondition*, *beginsWhen*, *endsWhen* and *postConditions* are gathered in UMM by the business collaboration protocol use case.

Exactly two roles participate in a binary collaboration. In our example these roles are a customer and a seller, which are defined by *Role* elements. The *nameID* of a role must be unique for all binary collaborations regardless the fact that a role with the same name exists in another binary collaboration. The next element *Start* maps the UMM transition from the initial pseudo state to the first business transaction activity, which is *Search Products*. This business transaction activity is referenced by name (*toBusinessState*) and id (*toBusinessStateIDREF*). The referenced business transaction activity must be in the following list of all business transaction activities of the binary collaboration. Thus, the next elements define the business transaction activity *search products* as well as *order products* and *register customer*. The definition of business transaction activities was explained in detail above.

Next, the *Success* element specifies the successful completion, i.e. the transition from *Order Products* to the successful end state. The last business transaction activity *Order Products* is referenced by name (*from BusinessState*) and id (*fromBusinessStateIDREF*). The value *BusinessSuccess* for *conditionGuard* of the transition is taken from an enumerated list of different success and failure types. Similarly, the *Failure* elements specify the transitions from *Register Customer* and from *Order Products* to the failing end state.

Only then all the other transitions between the business activities are declared. A transition has its own id and references both source and target business transaction activity by name and id. Again a transition guard from the enumerated list of success and failure types might be specified in *conditionGuard.* In addition the child element *ConditionExpression* enables a more complex specification reflecting the UMM business entity states. In our example we use an OCL syntax to denote the guards on the transitions. The Decision element - which follows the transition declarations - is the equivalent to the UML decision (depicted as rhombus). The underlying decision must be given in the subelement *ConditionExpression*; e.g. the OCL statement on whether the customer information is in state *Confirmed* or not. Other BPSS elements - not used in our example - include *Fork* for alternative flows and *Join* for parallel flows. Transitions reference not only business transaction activities as source and target, but also *Fork*, *Join*, and *Decision*. For example, the first transition statement is from Search Product to the decision node guarded by the fact that products are in state *Found* and substate *Wanted.*

UMM business collaboration protocols are always built by business transaction activities and do not include collaboration activities. These might be used in BPSS only. A collaboration activity references another binary collaboration which thereby becomes

part of the parent binary collaboration. The other attributes of the collaboration activity are the same as for the business transaction activity except that *timeToPerform* and *isConcurrent* are not appropriate. The *fromRoleIDREF* of the collaboration activity binds to the *initiatingRole* of the included binary collaboration. Note, that the id values must be different and the names of the roles might be different. This allows to bind e.g. a buyer role from the parent collaboration to a customer role of the included collaboration.

```
<BinaryCollaboration name="SimplePurchaseManagement" nameID="BPSS-XML4BPM-BinaryCollaboration-001"
initiatingRoleIDREF="BPSS-XML4BPM-Role-001" isInnerCollaboration="false" pattern="nopatterns exist yet"
timeToPerform="P1D" preCondition="some text" beginsWhen="some text" endsWhen="some text"
postCondition="some text">
    <Role name="customer" nameID="BPSS-XML4BPM-Role-001"/>
    <Role name="seller" nameID="BPSS-XML4BPM-Role-002"/>
    <Start nameID="BPSS-XML4BPM-Start-001" toBusinessState="SearchProducts"
    toBusinessStateIDREF="BPSS-XML4BPM-Activity-002"/>
    <BusinessTransactionActivity name="SearchProducts" nameID="BPSS-XML4BPM-Activity-002" ... />
    <BusinessTransactionActivity name="OrderProducts" nameID="BPSS-XML4BPM-Activity-003" ... />
    <BusinessTransactionActivity name="RegisterCustomer" nameID="BPSS-XML4BPM-Activity-001"... />
    <Success nameID="BPSS-XML4BPM-Success-002" fromBusinessState="OrderProducts"
    fromBusinessStateIDREF="BPSS-XML4BPM-Activity-003" conditionGuard="BusinessSuccess"/>
    <Failure nameID="..." fromBusinessState="OrderProducts" ... conditionGuard="Failure"/>
    <Failure ... fromBusinessState="RegisterCustomer" ... conditionGuard="Failure"/>
    <Transition nameID="Transition-002-F001" fromBusinessState="SearchProduct"
    fromBusinessStateIDREF="BPSS-XML4BPM-Activity-002" toBusinessState="Decision1"
    toBusinessStateIDREF="BPSS-XML4BPM-Decision-001" conditionGuard="BusinessSuccess">
        <ConditionExpression expressionLanguage="OCL"
        expression="Products.oclInState(Found::Wanted) = TRUE"/>
    </Transition>
    <Transition ... fromBusinessState="Decision1" ... toBusinessState="RegisterCustomer" ...>
        <ConditionExpression ... expression="CustomerInformation.oclInState(Confirmed) = FALSE"/>
    </Transition>
    <Transition ... fromBusinessState="Decision1" ... toBusinessState="OrderProducts" ... >
        <ConditionExpression ... expression="CustomerInformation.oclInState(Confirmed) = TRUE"/>
    </Transition>
    <Transition ... fromBusinessState="RegisterCustomer" ... toBusinessState="OrderProducts" ...
    ConditionGuard="BusinessSuccess">
        <ConditionExpression ... expression="CustomerInformation.oclInState(Confirmed) = TRUE"/>
    </Transition>
    <Decision name="Decision1" nameID="BPSS-XML4BPM-Decision-001">
            <ConditionExpression ... expression="CustomerInformation.oclInState(Confirmed)"/>
    </Decision>
</BinaryCollaboration>
```

3.4 Mulitparty Transaction

The current version of BPSS 1.10 uses binary collaborations only. The XML schema allows the specification of multiparty transaction, but it is not recommended to use them. The concepts for multiparty collaborations might change considerably within the next revision. Multiparty collaborations are always synthesized by binary collaborations. However, the multiparty statement does not list the included binary collaborations. Instead the multiparty statement list the business partner roles that participate in the multiparty collaboration. Each business partner role includes one or more *Performs* elements to bind roles of binary collaborations. Since the id of a role is unique for all binary collaborations, the binary collaborations are also unambiguously identified.

The business partners in our example *simple order management* are buyer, seller and bank. The buyer takes on the role of customer in the simple purchase management. The seller binds the role of seller in the simple purchase management and the role of the credibility requestor in check credibility. The bank supports the role of a bank in check credibility. Furthermore, transitions between business transaction activities of different binary collaborations are supported. Since the business transaction activities of our example are nested this concept is not used in the code fragment below.

```
<MultiPartyCollaboration name="SimpleOrderManagement" nameID="BPSS-XML4BPM-MultiParty-001">
    <BusinessPartnerRole name="buyer" nameID="BPSS-XML4BPM-MultiRole-001">
        <Performs nameID="" role="customer" roleIDREF="BPSS-XML4BPM-Role-001"/>
    </BusinessPartnerRole>
    <BusinessPartnerRole nameID="BPSS-XML4BPM-MultiRole-002" name="seller">
        <Performs nameID="" role="seller" roleIDREF="BPSS-XML4BPM-Role-002"/>
        <Performs nameID="" role="credibilityRequestor" roleIDREF="BPSS-XML4BPM-Role-003"/>
    </BusinessPartnerRole>
    <BusinessPartnerRole nameID="" name="bank">
        <Performs nameID="" role="bank" roleIDREF="BPSS-XML4BPM-Role-004"/>
    </BusinessPartnerRole>
</MultiPartyCollaboration>
```

4 Summary

The BPSS specification states that its goal is to provide the bridge between e-business process modeling and specification of e-business software components. BPSS does not require any particular e-business process modeling methodology. Nevertheless, main concepts of BPSS are based on UMM, or better its meta model. Thus it is close at hand to model e-business collaborations with UMM and to map these models to XML-based BPSS. The gap between UMM and BPSS is quite close. This approach enables e-business software to interpret the choreography specified by UMM models.

In this paper we first presented UMM as a top-down approach. Apart from introducing UMM's methodology, this section helps to understand the background of many BPSS concepts. Then the UMM business collaboration models are mapped to BPSS. We explained the mapping the other way round by a bottom-up approach. It is demonstrated that most of the UMM semantics - except those for requirements gathering - result in a equivalent BPSS counterpart. Most of them use the same terms and structures. Only very few UMM concepts are not supported by BPSS. Additional concepts available in BPSS but not supported in UMM are rare exceptions.

From the arguments above it becomes obvious that alignment of BPSS and UMM is desirable. However, BPSS must also be aligned with the other ebXML specifications. The BPSS team has done a good job to deal with these sometimes conflicting interests. At the end of the 18-month ebXML initiative UN/CEFACT and OASIS agreed that ebXML business process specifications are maintained by UN/CEFACT. UN/CEFACT has signaled steps toward UMM BRV alignment in BPSS 2.0 and towards BDV alignment in BPSS 3.0. OASIS has formed its new ebXML business process technical committee in October 2003. Therefore, we stick with the current version 1.10 and await BPSS's future.

References

[BJR98] Booch, G., Jacobson, I., Rumbaugh J.: The Unified Modeling Language User Guide. Addison Wesley Object Technology Series, Reading, (1998)

[HHK02] Hofreiter, B., Huemer, C., Klas, W.: ebXML: Status, Research Issues and Obstacles. Proc. of 12th Int. Workshop on Research Issues on Data Engineering (RIDE02), San Jose (2002)

[HHN04] Hofreiter, B., Huemer, C., Naujok, K.-D.: UN/CEFACT's Business Collaboration Framework - Motivation and Basic Concepts. Proc. of MKWI'04 Track on Co-ordination in Value Creation networks / Agent Technology for Business Applications, LNI GITO (2004)

[HH03] Hofreiter, B., Huemer, C.: Modeling Business Collaborations in Context. Proc. of On The Move to Meaningful Internet Systems 2003: OTM 2003 Workshops, Springer LNCS, Catania (2003)

[ISO95] ISO: Open-edi Reference Model. ISO/IEC JTC 1/SC30 ISO Standard 14662 (1995)

[UN03a] UN/CEFACT TMG: UN/CEFACT Modelling Methodology - Meta Model, Revision 12. (2003), http://www.untmg.org/downloads/General/approved/UMM-MM-V20030117.zip

[UN03b] UN/CEFACT TMG: UMM User Guide, Revision 12. (2003) http://www.untmg.org/downloads/General/approved/UMM-UG-V20030922.zip

[UN03c] UN/CEFACT TMG: UN/CEFACT – ebXML Business Process Specification Schema, Version 1.10, http://www.untmg.org/downloads/General/approved/ebBPSS-v1pt10.zip

[UN03d] UN/CEFACT TMG: Core Components Technical Specification – Part 8 of the ebXML Framework, Version 2.01, (2003), http://www.untmg.org/downloads/General/approved/CEFACT-CCTS-Version-2pt01.zip

[UO01] UN/CEFACT, OASIS: ebXML Technical Architecture Specification v1.0.4. (2001), http://www.ebxml.org/specs/ebTA.pdf

Exchanging Business Process Models with GXL

Andreas Winter
Institute for Software Technology
University of Koblenz
D-56070 Koblenz
Universitätsstraße 1
www.uni-koblenz.de/~winter/
mailto:winter@uni-koblenz.de

Carlo Simon
Institute for Management
University of Koblenz
D-56070 Koblenz
Universitätsstraße 1
www.uni-koblenz.de/~simon/
mailto:simon@uni-koblenz.de

Abstract: GXL (Graph eXchange Language) is an XML-based standard exchange language for sharing graph data between tools. GXL can be customized to exchange application specific types of graphs. This is done by exchanging both, the instance graph, representing the data itself, and the schema, representing the graph structure.

Business Process Models are usually depicted in a graph-like form. So, GXL is also a proper means to exchange those data. This paper shows, how to customize GXL in order to exchange business process models depicted as Event-driven Process Chains or Workflow-Nets as examples for the control flow part of business process models. Each level of modeling is exemplarily demonstrated from the meta schemas down to instances of graphs.

1 Introduction

Graphs are widely used for representing and analysing structured data in various areas. They combine visual descriptiveness and clearness with mathematical foundation leading to efficient data-structures and -algorithms. Thus, a great variety of tools relies on various kinds of graph based structures. Offering a generally applicable means for *exchanging graph based structures* provides a broad basis for data-exchange.

A general and widely accepted interchange format for graphs has to fulfill various demands (cf. [Mü98], [KGW98]).

Adaptability: Different problems solved by graph-based tools require different problem-related graph models. Graph-based tools might base e. g. on trees, directed or undirected graphs, node and edge attributed graphs, node and edge typed graphs, hypergraphs, or hierarchical graphs, or combinations of these. A standard graph exchange language would need to be flexible enough to be an intermediary for these and other graph models.

Exchanging graphs requires to agree on the kind of graphs to be interchanged. An exchange language has to offer means to define and customize graph structure to certain problem domains by defining e. g. node- and edge types, incidence relations between node- and edge types, and multiplicity constraints.

Processability: exchanged data has to be processed, efficiently. The efficiency of exchange formats refers to the exchange process rather than the efficient usage of that data in

certain applications [Bl04]. Thus, graph documents have to be generated from stored data, easily, have to be transferred between interoperating tools, rapidly, and have to be imported into current applications, easily. Furthermore, a general graph exchange language should come along with a set of tools supporting its usage.

Distribution: standard exchange formats are only useful, if they are supported by a large number of tools. A graph exchange language will be successful, if it is supported by various components e. g. for generating graphs, for analyzing graphs and applying graph algorithms and graph transformations, and for visualizing graphs.

The *GXL Graph eXchange language* [HWS00], [Wi02] was developed to offer such a standard interchange format for graphs complying these demands:

Adaptability: GXL represents *TGraphs*, i. e. (node and edge) typed, (node and edge) attributed, ordered, and directed Graphs [EWD+96] which are extended to represent *hierarchical graphs* [Bu01] and *hypergraphs* [Be76]. Typed, attributed, ordered, and directed, hierarchical graphs and hypergraphs form a general graph model which covers most of usually used graph structures. Thus, GXL is able to exchange graphs following a wide range of graph models.

GXL supports both, exchanging graphs (instance graph) and graph structure (graph schema). Class diagrams are a proper way to define graph structures. Since these information can easily be mapped to graphs [Wi02], GXL exchanges instance and schema information by using the same mechanisms.

Processability: GXL is defined as an *XML language* [W3C00]. Graphs and schemas are exchanged by XML streams following the GXL document type definition [Wi02] or the GXL XML Schema specification [GXLa] . GXL was defined plain and simple. So, the GXL DTD only requires 18 XML-elements to support a most general and powerfull graph model.

Like all XML languages, GXL gains profit from a large number of already existing XML-based tools. GXL base functionality can be implemented easily using XML standard tools like Xerces parser for reading or Xalan for manipulating GXL documents [Apache]. Furthermore, special GXL tools exist, e. g. for validating GXL documents [Ka03]. The efficiency of representing and processing GXL data is (only) restricted by the efficiency of XML.

Distribution: GXL was ratified as standard exchange format in software reengineering at the Dagstuhl Seminar on *Interoperability of Reengineering Tools* in January 2001 [EKM01]. GXL also defines the foundation of the GTXL exchange language for Graph transformation system [Ta01] [GTXL]. During the last years various groups in reengineering, graph transformation, graph drawing, and other areas of software engineering have added GXL support in their tools. A still growing list of currently more than 40 tools supporting GXL can be found at [GXLb].

Summarizing, GXL can be viewed as an adaptable, easily processable, and widely distributed standard exchange language for graphs.

GXL originally intended to become a commonly accepted exchange language for information on software systems providing interoperability of various reengineering tools. Since GXL was developed as a general graph exchange language, its use was extended to many areas that deal with graph-structured data.

This paper demonstrates the use of GXL as a language to exchange *Business Process Models*. Within such models, the work performed within businesses is described and put into a structure. Additionally, the binding of resources like humans or machines to specific activities is expressed. Business processes are usually depicted using graphical languages. In this paper we focuss on languages modeling dynamic aspects of business processes. These languages include Event-driven Process Chains [Sc94b] and Petri-Net based approaches like Workflow-Nets [Aal96]. For exchanging process models among different modeling environments and Workflow Management Systems, formal exchange languages are required.

Since Event-driven Process Chains and Petri-Net diagrams provide structural information, they are typically viewed as graphs. Thus, GXL as a graph exchange language is a proper means for exchanging such models. The reminder of this paper introduces the foundation of GXL (cf. section 2) and shows how GXL is customized to exchange Event-driven Process Chains (cf. section 3.1) and Workflow-Nets (cf. section 3.2). We finish our paper with some concluding remarks.

2 GXL Graph eXchange Language

GXL is an XML based configurable exchange format for graphs. It can be adapted to a broad variety of different types of graphs and hypergraphs. The adaptability of GXL is based on *metaschemas* specifying the required graph structure.

Thus, GXL representations of graphs typically consist of two parts: in a first part, the actual *graph* is specified, in a second (optional) part the graph structure is defined in a *graph schema*. Section 2.1 explains, how graph data is represented in GXL streams. The definition and exchange of graph schemas as special GXL graphs, is presented in section 2.2. GXL exchanges graphs and schemas using *the same* type of XML documents.

2.1 Exchanging Graphs with GXL

GXL supports the exchange of directed or undirected graphs consisting of typed, attributed, and ordered nodes, edges and hyperedges, incidences, and hierarchical subgraphs.

As a simple example, Figure 1 depicts a fragment of a London map used in the Scotland-Yard game [ScY]. The map shows some sights and connection points for changing public transport by taxi, bus, or subway (tube). A graph representation of that map is done by an *undirected, node- and edge typed, node attributed graph* depicted as UML object diagram.

The graph consists of two different kinds of nodes. Nodes of class Junction depict street-crossings, where public transport can be changed and sight-nodes emphasize sightseeing highlights like monuments or squares. Both, Junction- and Sight-nodes are identified by a number. Additionally, Sight-nodes carry the sights' name. It is possible to navigate between crossings and sights by taxi, bus, or tube. These connections are modeled by taxi-, bus-, and tube-edges.

GXL provides constructs for representing and exchanging graphs such as the one in Figure 1. Here constructs are required to represent nodes, edges, incidences, node- and edge-classes, and node-attributes.

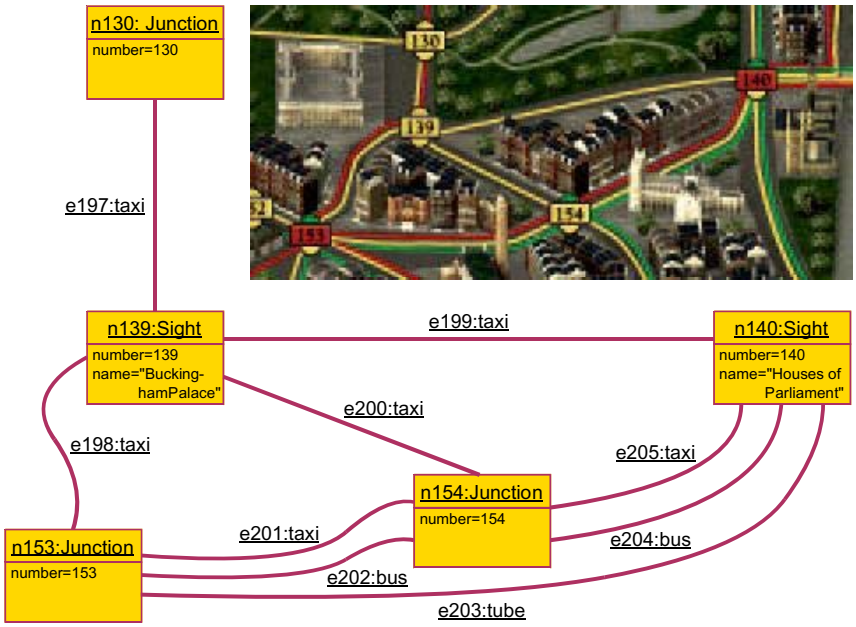


Figure 1: undirected typed, attributed graph

Figure 2 depicts the graph from Figure 1 as an XML document according to the GXL structure. The first two lines specify the used XML version and refer to the GXL document type definition (gxl-1.0.dtd). The GXL DTD can be found at [Wi02] and an XML-schema version is given at [GXLa].

```
1  <?xml version="1.0"?>
2  <!DOCTYPE gxl SYSTEM "gxl-1.0.dtd">
3  <gxl xmlns:xlink="http://www.w3.org/1999/xlink">
4  <graph id="londonmap" edgeids="true" edgemode="undirected">
5    <type xlink:href="map.gxl#mapSchema"/>
6    <node id="n130"><type xlink:href="map.gxl#Junction"/>
7      <attr name="number"><int>130</int></attr></node>
8    <node id="n153"><type xlink:href="map.gxl#Junction"/>
9      <attr name="number"><int>153</int></attr></node>
10   <node id="n154"><type xlink:href="map.gxl#Junction"/>
11     <attr name="number"><int>154</int></attr></node>
12   <node id="n139"><type xlink:href="map.gxl#Sight"/>
13     <attr name="number"><int>139</int></attr>
14     <attr name="name"><string>Buckingham Palace</string></attr></node>
15   <node id="n140"><type xlink:href="map.gxl#Sight"/>
16     <attr name="number"><int>140</int></attr>
17     <attr name="name"><string>Houses of Parliament</string></attr></node>
18   <edge id="e197" from="n130" to="n139"><type xlink:href="map.gxl#taxi"/></edge>
19   <edge id="e198" from="n139" to="n153"><type xlink:href="map.gxl#taxi"/></edge>
20   <edge id="e199" from="n139" to="n140"><type xlink:href="map.gxl#taxi"/></edge>
21   <edge id="e200" from="n139" to="n154"><type xlink:href="map.gxl#taxi"/></edge>
22   <edge id="e201" from="n153" to="n154"><type xlink:href="map.gxl#taxi"/></edge>
23   <edge id="e205" from="n154" to="n140"><type xlink:href="map.gxl#taxi"/></edge>
24   <edge id="e202" from="n153" to="n154"><type xlink:href="map.gxl#bus"/></edge>
25   <edge id="e204" from="n154" to="n140"><type xlink:href="map.gxl#bus"/></edge>
26   <edge id="e203" from="n153" to="n140"><type xlink:href="map.gxl#tube"/></edge>
27 </graph>
28 </gxl>
```

Figure 2: GXL representation of Figure 1

The body of the document is enclosed in <**gxl**> elements. Line 3 includes the xlink-namespace, which is required for referring to external XML documents. Graphs are enclosed in <**graph**> elements. Line 4 introduces the graph as londonmap, specifies that edges must have identifiers, and that the graph has to be interpreted as an undirected graph. Nodes and edges are represented by <**node**> and <**edge**> elements. Both, nodes and edges can be located by their id-attribute. Incidences of edges are stored in from and to attributes within <**edge**> tags. These attributes refer to the id's of the adjacent nodes. For undirected graphs, like the one in figure 1, the implicitly given edge-orientation by from and to-attributes has to be ignored. In the case of directed graphs, from and to refer to nodes representing origin and end of an edge.

<**node**> and <**edge**> elements may contain further attribute information, stored in <**attr**> elements. <**attr**> elements describe attribute names and values. Like OCL [WK98], GXL provides <**bool**>, <**int**>, <**float**>, and <**string**> attributes. Furthermore, enumeration values (<**enum**>) and URI-references (<**locator**>) to externally stored objects are supported. Attributes in GXL might also be structured. Here, GXL offers composite attributes like sequences (<**seq**>), sets (<**set**>), multi sets (<**bag**>), and tuples (<**tup**>).

Graphs and graph elements may refer to the according schema information, stored in <**type**> elements. Using xlink [W3C01] <**type**> elements refer to a GXL-document defining the schema (here map.gxl) and the appropriate node or edge class. An extract of the GXL stream for that schema is shown in Figure 5. The relation between GXL graphs and their according schema can be checked within the GXL framework using the GXL validator [Ka03].

In addition to the GXL features, described with Figure 2, GXL provides the exchange of hypergraphs and hierarchical graphs. More information on exchanging various kinds of graphs can be found at [Wi02].

2.2 Exchanging Graphclasses with GXL

Graphs like the one depicted in Figure 1 contain nodes representing crossings and sights. They are connected by three different kinds of edges representing taxi, bus and subway connections. Exchanging graphs like this, requires to know about that structure. A graph schema, defining the structure of graphs like the one in Figure 1 is shown in Figure 3. The graph schema is depicted as an UML class diagram [BRJ99], where classes define node classes and associations define edge classes.

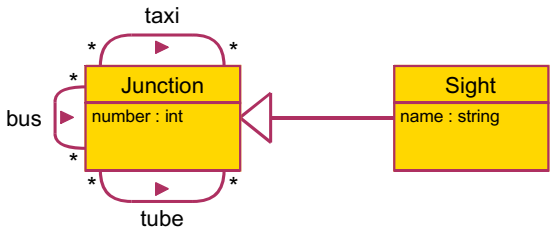


Figure 3: Graph schema

The graph schema defines node class Junction and its specialization Sight for modeling crossings and sights. Nodes of class Junction are attributed with their number and addi-

tionally nodes of class Sight carry a name attribute. All nodes may be connected by edges of edge class taxi-, bus-, and tube.

Since class diagrams are structured information themselves, they can be represented as graphs as well. For exchanging graph schemas, GXL uses a predefined way to translate schemas into graphs. Graphs representing schema information follow the *GXL Meta-schema* [Wi02]. A graph representation of the schema in Figure 3 is depicted in Figure 4.

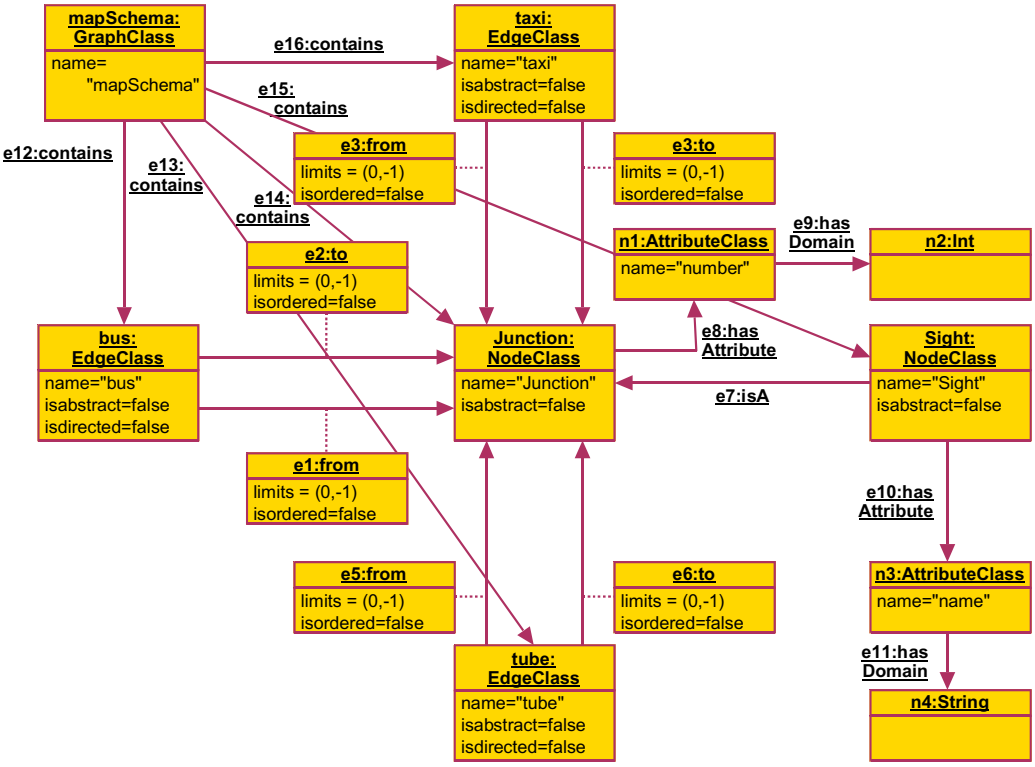


Figure 4: Graph schema represented as Graph

Node classes and edge classes are modeled by NodeClass-nodes and EdgeClass-nodes. Their name-attributes determine the node and edge classes' names. For example, node class Junction is represented by the NodeClass node with OID `Junction` and edge class taxi is depicted by the EdgeClass node with OID `taxi`. from and to-edges connect EdgeClasses with the incident NodeClasses. Their attributes contain multiplicities (limits) and indicate if the incidence should be treated as ordered (isordered). Attribute information is modeled by AttributeClass nodes. They can be connected to both, node and edge classes by hasAttribute edges. hasDomain edges link to attribute domains (Int or String nodes). GXL provides generalization of node and edge classes by isA-edges; cf. edge *e7* connecting `Sight:NodeClass` and `Junction:NodeClass`. isabstract=false marks concrete classes and isdirected=false labels undirected edge classes. The complete graph class is represented by a GraphClass node which collects its node and edge classes by contains edges.

Graph class mapSchema in Figure 4 contains node classes Junction and Sight and edge classes taxi, bus, and tube. GXL instances matching the Map schema refer to these nodes as schema references, e. g. node `154` in Figure 2 (line 10) refers to node `Junction` in the corresponding schema identifying `n154` as a street crossing.

```
1  <?xml version="1.0"?>
2  <!DOCTYPE gxl SYSTEM "http://www.gupro.de/GXL/gxl-1.0.dtd">
3  <gxl xmlns:xlink="http://www.w3.org/1999/xlink">
4  <graph id="map" edgeids="true" edgemode="directed">
5    <type xlink:href="http://www.gupro.de/GXL/gxl-1.0.gxl#gxl-1.0"/>
6    <node id="mapSchema">
7      <type xlink:href="http://www.gupro.de/GXL/gxl-1.0.gxl#GraphClass"/>
8      <attr name="name"><string>mapSchema</string></attr></node>
9    <node id="Junction">
10     <type xlink:href="http://www.gupro.de/GXL/gxl-1.0.gxl#NodeClass"/>
11     <attr name="name"><string>Junction</string></attr>
12     <attr name="isabstract"><bool>false</bool></attr></node>
13 ...
14   <node id="taxi">
15     <type xlink:href="http://www.gupro.de/GXL/gxl-1.0.gxl#EdgeClass"/>
16     <attr name="name"><string>taxi</string></attr>
17     <attr name="isabstract"><bool>false</bool></attr>
18     <attr name="isdirected"><bool>false</bool></attr></node>
19 ...
20   <node id="n1">
21     <type xlink:href="http://www.gupro.de/GXL/gxl-1.0.gxl#AttributeClass"/>
22     <attr name="name"><string>number</string></attr></node>
23   <node id="n2">
24     <type xlink:href="http://www.gupro.de/GXL/gxl-1.0.gxl#Int"/></node>
25 ...
26   <edge id="e3" from="taxi" to="Junction">
27     <type xlink:href="http://www.gupro.de/GXL/gxl-1.0.gxl#from"/>
28     <attr name="limits"><tup><int>0</int><int>−1</int></tup></attr>
29     <attr name="isordered"><bool>false</bool></attr></edge>
30   <edge id="e4" from="taxi" to="Junction">
31     <type xlink:href="http://www.gupro.de/GXL/gxl-1.0.gxl#to"/>
32     <attr name="limits"><tup><int>0</int><int>−1</int></tup></attr>
33     <attr name="isordered"><bool>false</bool></attr></edge>
34 ...
35   <edge id="e8" from="Junction" to="n1">
36     <type xlink:href="http://www.gupro.de/GXL/gxl-1.0.gxl#hasAttribute"/>
37   </edge>
38   <edge id="e9" from="n1" to="n2">
39     <type xlink:href="http://www.gupro.de/GXL/gxl-1.0.gxl#hasDomain"/></edge>
40 ...
41   <edge id="e14" from="mapSchema" to="Junction">
42     <type xlink:href="http://www.gupro.de/GXL/gxl-1.0.gxl#contains"/></edge>
43 ...
44 </graph>
45 </gxl>
```

Figure 5: GXL representation of schema graph in Figure 4 (extract)

Like all GXL graphs, this graph can be stored as an XML-stream following the GXL definition. An extract of that GXL document is depicted in figure 5. Graphs, modeling schema information, are instances of the GXL-Metaschema whose GXL representation is stored at `http://www.gupro.de/GXL/gxl-1.0.gxl`. Thus, all schema references link to nodes of that file: e. g. NodeClass node `Junction` refers to the definition of its type in the GXL-Metaschema (line 10). Analogously, lines 14-18 show the definition of edgeClass `taxi`. taxi edges connect Junction nodes. These incidences are modeled by edges `e3` and `e4` (lines 26-33). Attribute structures are represented by AttributeClass and domain nodes. Figure 5 shows the attribute structure associated to node class Junction in lines 20-24 (Attribute and Domain) and in lines 35-39 (connections). Each GXL document, defining a graph schema, possesses at least one GraphClass node representing the

graph class. Lines 6-8 denote the GraphClass-node for the map graph class. It is connected to all of its node- and edge classes (e. g. lines 41f).
Since the GXL Metaschema is a schema, it is represented by an GXL document referring to the GXL Metaschema, namely itself. The GXL Metaschema, its representation as UML-class diagram and as GXL document is documented at [GXLa].

2.3 Customizing GXL

The previous sections shortly introduced GXL for representing graph data and associated graph schemas. Both, graph instances and graph schemas are exchanged by the same type of XML documents following the GXL specification.
Using GXL for exchanging graph data in a particular application domain requires to constrain the form of graphs, i. e. limiting the types of nodes and edges. GXL is customized by defining graph schemas. These schemas determine

- which node, edge, and hyperedge classes (types) can be used,
- which relations can exist between nodes, edges, and hyperedges of given classes,
- which attributes can be associated with nodes, edges, and hyperedges,
- which graph hierarchies are supported, and
- which additional constraints (such as ordering of incidences, degree restrictions) have to be imposed.

These constraints specialize the graph structure to represent the domain of interest. It is useful to standardize graph structures for particular domains by *GXL Reference Schemas*. Currently, those GXL reference schemas are defined for various reverses engineering domains e. g. [LTP04] defines a GXL reference schema for language independent representation of source code data and [FSH+01, FB02] deal with the definition of a GXL reference schema for C++-abstract syntax trees.
Tailoring GXL for exchanging *Business Process Models* requires to specify *GXL Schemas for Business Process Models*. Depending on the used modeling approach and notation, these schemas define the relevant modeling concepts and their associations. The following section introduces candidates for those GXL schemas for customizing GXL to exchange *Event-driven Process Chains* and *Workflow-Nets* as a variant of Petri-Nets.

3 Exchanging Business Process Models

Business process models must provide the following views on organizations: control flow, information, and organizational structure [Sc94b],[JBS97, p 98ff]. In this paper we focus on the control flow part, which is modeled either in Event-driven Process Chains or in Petri-Nets. All other aspects of business process models can be treated analogeously.
Several notations have been introduced which all profit from the ability of Petri-Nets to represent both the actual process as well as business resources like humans or information. A brought overview over published approaches to modeling business processes with Petri-Nets is given by Janssens, Verelst and Weyn in [JVW00]. Beside the descriptive nature of models and the ability to verify the models with standard Petri net analysing techniques or against special process specifications [Si02b, Si02a], their automatic execution within a Workflow Management System [WfM96] is one of their aims.

Both the variety of description languages as well as the necessity to execute the models in Workflow Management Systems require formal languages that can be used to exchange models among different platforms. In the following we demonstrate how to use GXL as such a language.

To illustrate our approach, we have have chosen Event-driven Process Chains (EPCs) introduced by Scheer [Sc94b] and Workflow-Nets (WF-Nets) introduced by van der Aalst [Aal96, AH02] as exemplary Petri-Net-based notations for business process modeling. We are defining a meta schema for each of these notations and demonstrate the instantiation of these models by examples.

3.1 Exchanging Event-driven Process Chains with GXL

Event-driven Process Chains (EPCs) [KNS92] are one central concept within the ARIS business process framework [Sc00] basing on stochastic networks and Petri-Nets.

Events (depicted by hexagons) and functions (depicted by ovals) are the basic elements of EPCs which are connected by a flow relation and additional connectives used to span complex process structures. Figure 6 shows the basic elements of EPCs as defined in [Sc94a].

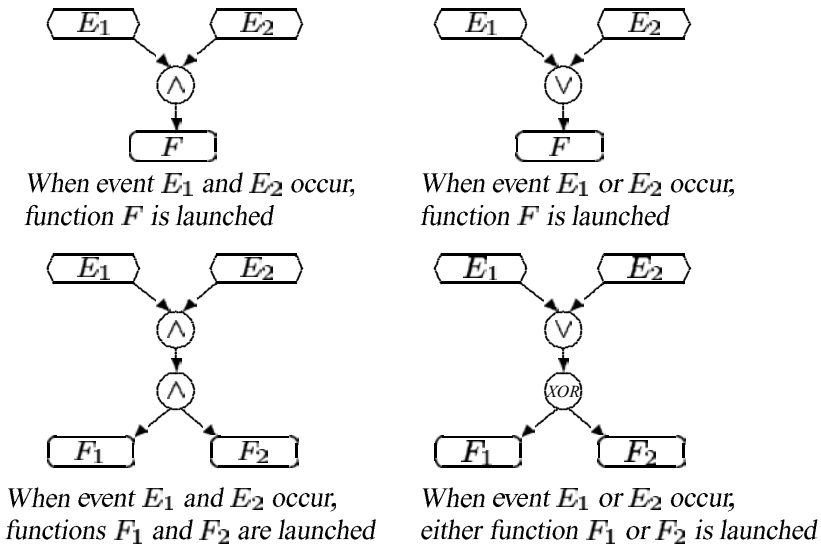


Figure 6: Event relationships in EPC from [Sc94a]

EPCs support branching and merging control flows. Logical operators ($\wedge$, $\vee$, xor) depicted in circles, indicate how incoming events are combined to enable function execution, how control flow is shared between concurrent functions, and how different functions result in an event. Furthermore, EPCs provide sequences of control flow operators. In general, control flows in EPCs span bipartite (hyper)-graphs, where functions follow on events and events follow on functions.

Syntax and semantics of EPCs is explained somehow vague. Various approaches exist to resolve this uncertainness by specifying the concrete syntax (e. g. [St99]) and semantics (e. g. [NR02], [ADK02]) of EPC. In the following, we will not address that problem, we only specify the EPC-syntax as far as it is required to explain the idea of using GXL for exchanging EPC-based process models and assume an intuitive EPC semantics.

3.1.1 Metaschema for Event-driven Process Chains

Adapting GXL to exchange EPCs requires the specification of a *GXL schema for Event-driven Process Chains*. This schema has to cover all modeling concepts and their associations used in EPC. In [Sc00] all modeling techniques used within the ARIS business process framework are introduced along their metaschemas. The EPC meta schema in [Sc00, p 128] was designed to roughly describe modeling techniques and their relationships to other modeling techniques. Although this model is sound within the ARIS business process framework, it is not appropriate for GXL exchange and has to be adapted. Especially, it does not cover different kinds of control flows between events and functions.

A usefull meta schema for defining the exchange of EPC models has to specify the complete syntax of EPC. It has to cover concepts for modeling *events*, *functions* and the various *control flows*. Figure 7 shows the class diagram part of a GXL schema for EPC. To explain the idea of exchanging business process models with GXL, this schema is restricted to those EPC concepts described in this paper. A complete GXL schema for EPC should also contain concepts to exchange conditions, messages and hierarchies.

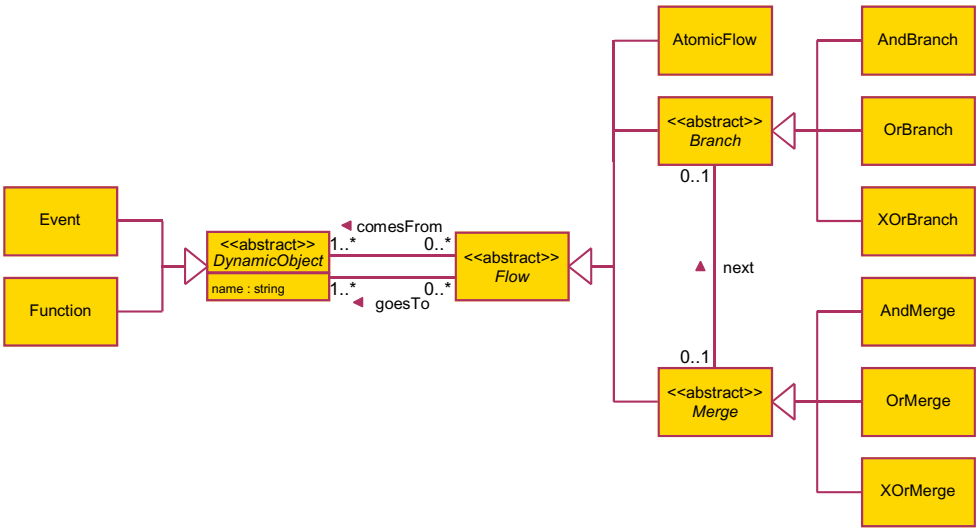


Figure 7: GXL schema for Event-driven Process Chains

Events and Functions are connected by Flows. comesFrom and goesTo-edges express the directed flow of control between Events, Functions and Flows. Flows are distinguished in AtomicFlows linking from one event to one function or vice versa, in Branches branching control flow to various events or functions, and in Merges joining control flow from various events or functions. Branches and Merges occur in their $\wedge$, $\vee$, and xor-variants. Direct sequences of flows between merges and branches (cf. Figure 6) are modeled by next-edges.

Further constraints add syntactical restrictions to a metaschema. For instance, these constraints ensure, that functions and events alternate on the control flow, that branches have one incoming and many outgoing connections, and that merges have many incoming and only one outgoing connection. These constraints are not required to exchange graphs with GXL, but they are mandatory to decide if an EPC model conforms its syntax definition. Complete meta schemas for EPC with their class diagram and constraint part are given in [Wi00, 192]. These schemas also include hierarchies of EPC and the embedding of EPC in multi-perspective modeling environments.

A meta schema like the one in Figure 7 controls the exchange of EPC business process models via GXL. According the GXL meta schema the EPC schema is exchanged by an suitable GXL graph (cf. section 2.2).

3.1.2 Exemplary EPC in GXL

Exchanging an EPC with GXL is based on *translating* the EPC diagram into an GXL graph according the metaschema given in Figure 7. In the following, we will demonstate this with an example EPC modeling a business process for processing customer orders [BH99, p. 62] in Figure 8.

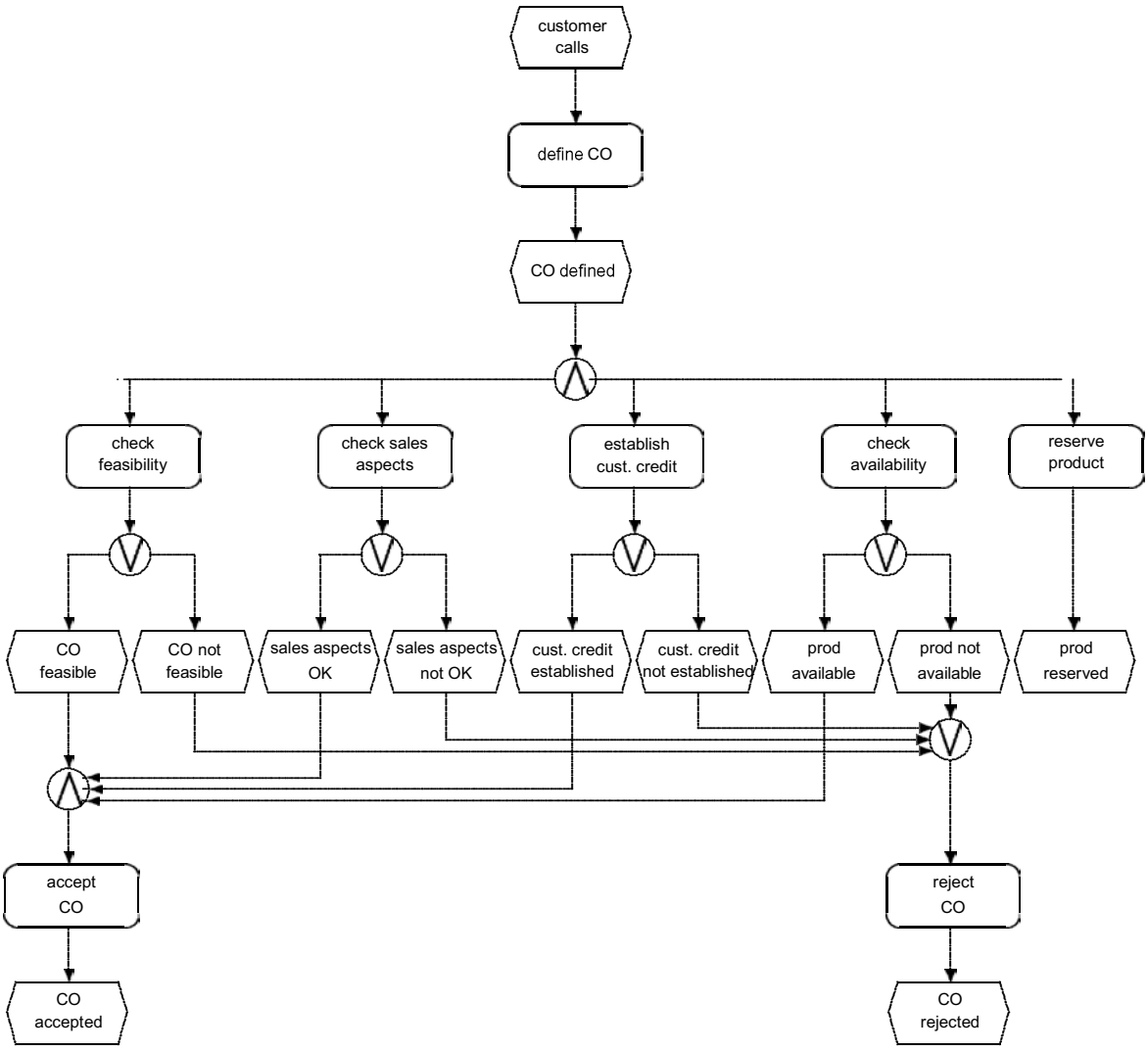


Figure 8: An exemplary EPC (from [BH99, p. 62])

Business process "*process customer order*" starts with a customers call. After defining the order, control flow branches into concurrent activities which perform various tests. Depending on their results, the customers order is either accepted or rejected.

Figure 9 shows an extract of the GXL graph representing the EPC depicted in Figure 8. Events are modeled by Event and Functions by Function nodes: e. g. node `ne1:Event` represents the event *customer calls* starting the business process and `nf1:Function` depicts the first function, which defines the customer order (*define CO*).

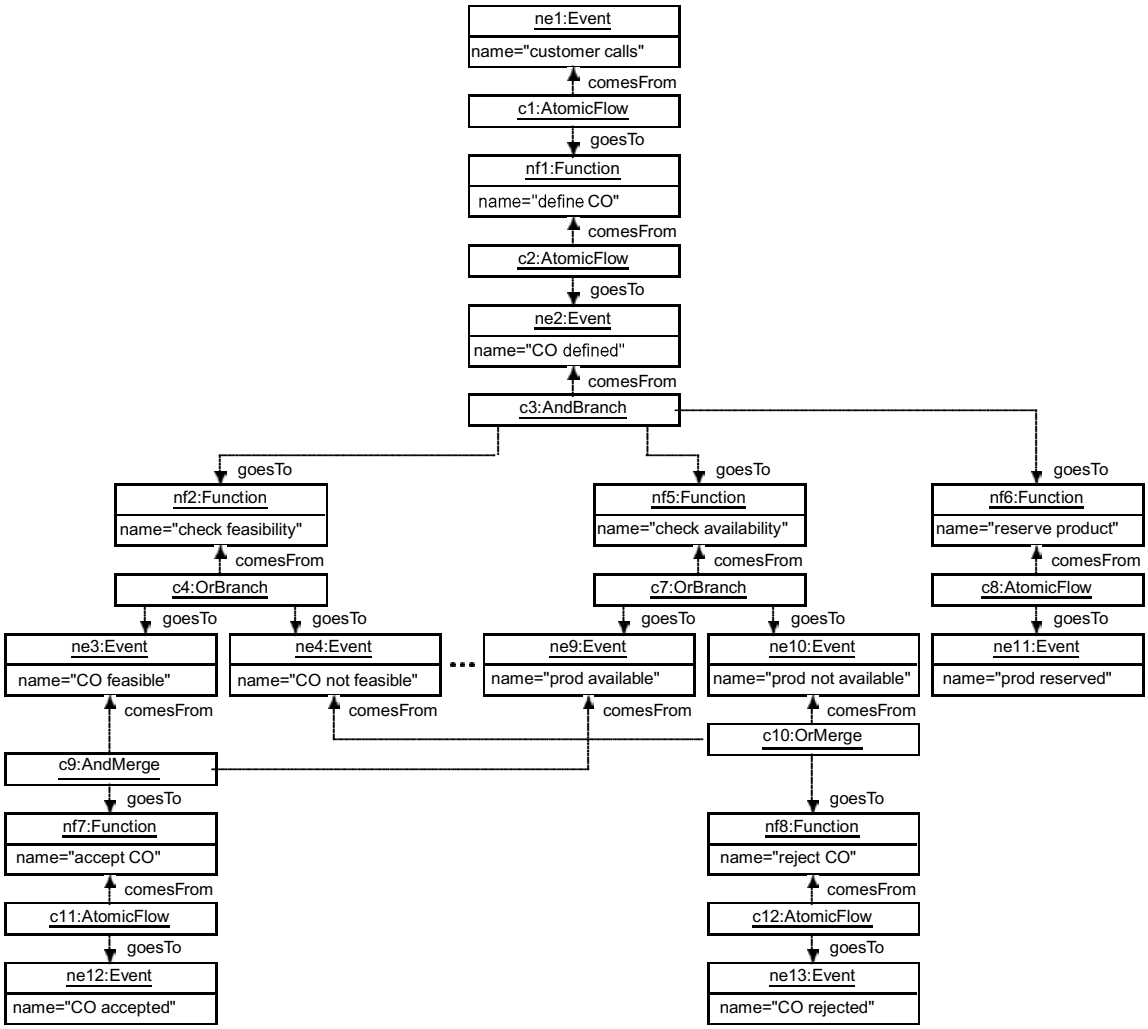


Figure 9: Representation of the exemplary EPC of figure 8 as a UML instance diagram

On the contrary to the EPC diagram in Figure 8, where atomic flows of control are depicted as directed edges, the EPC meta schema demands modeling atomic flows by nodes. Analogously to merges and branches, AtomicFlow nodes connect associated events or functions by comesFrom and goesTo edges (cf. node `c1:AtomicFlow`).

Branching the flow of control into various concurrent activities is modeled by AndBranch nodes. For instance node `c3:AndBranch` links the (incoming) event *CO defined* with the (outgoing) concurrent functions by comesFrom and goesTo edges. Analogously, AndMerge and OrMerge nodes collect control flows. E. g. node `c9:AndMerge` conjugates the events *CO feasible* and *prod available* and proceeds with function *accept CO*.

Figure 10 shows the GXL stream representing an extract of the graph in Figure 9. This GXL document refers to the GXL representation of the EPC meta schema (cf. Figure 7) in line 5. Events, Functions and Flows are stored by suitable <**node**> elements (lines 9-16) and their connections are modeled by <**edge**> elements. Lines 21-26 show the conjunction (node `c9:AndMerge` in line 16) of events *CO feasible* (node ne3:Event) and *prod available* (node ne9:Event) leading to the execution of function *accept CO* (node nf7:Function).

```
1  <?xml version="1.0"?>
2  <!DOCTYPE gxl SYSTEM "http://www.gupro.de/GXL/gxl-1.0.dtd">
3  <gxl xmlns:xlink="http://www.w3.org/1999/xlink">
4  <graph id="exampleEPC" edgeids="true" edgemode="directed">
5    <type xlink:href="metaEPC.gxl#epcSchema"/>
6    <node id="ne1"><type xlink:href="epcSchema.gxl#Event"/>
7      <attr name="name"><string>customer calls</string></attr></node>
8  ...
9    <node id="nf1"><type xlink:href="epcSchema.gxl#Function"/>
10     <attr name="name"><string>define CO</string></attr></node>
11 ...
12   <node id="c1"><type xlink:href="epcSchema.gxl#AtomicFlow"/></node>
13 ...
14   <node id="c3"><type xlink:href="epcSchema.gxl#AndBranch"/></node>
15 ...
16   <node id="c9"><type xlink:href="epcSchema.gxl#AndMerge"/></node>
17 ...
18   <edge id="e1" from="c1" to="ne1">
19     <type xlink:href="epcSchema.gxl#comesFrom"/></edge>
20 ...
21   <edge id="e17" from="c9" to="ne3">
22     <type xlink:href="epcSchema.gxl#comesFrom"/></edge>
23   <edge id="e18" from="c9" to="ne9">
24     <type xlink:href="epcSchema.gxl#comesFrom"/></edge>
25   <edge id="e22" from="c9" to="nf7">
26     <type xlink:href="epcSchema.gxl#goesTo"/></edge>
27 ...
28 </graph>
29 </gxl>
```

Figure 10: GXL representation EPC graph in Figure 9 (extract)

3.2 Exchanging Workflow-Nets with GXL

Petri-Nets [Pe62, Ba96] are a formally sound and well understood approach to describe and analyse concurrent and non-deterministic processes.

Places represent system states or conditions and *transitions* model specified actions provoking the change of states. Usually places are depicted by circles and transitions by rectangles. Flows connect places and transitions. Like flows in EPC, they span a bipartite graph, where places proceed transitions and transitions proceed places. The base modeling constructs of those place-transition-nets is shown in Figure 11.

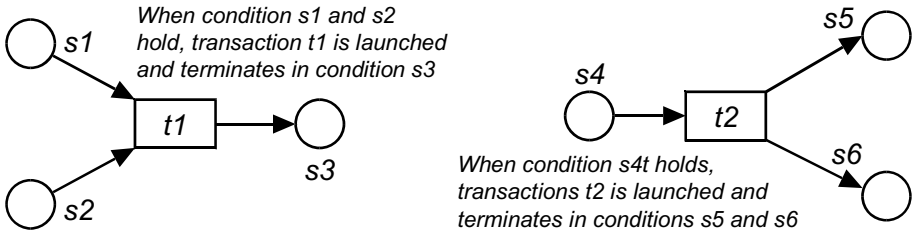


Figure 11: Petri-Net modeling constructs

Process modeling is supported by various different Petri-Net variants. To explain the exchange of business processes modeled by Petri-Nets we have chosen *Workflow-Nets* defined by van der Aalst [Aal96] as an example.

3.2.1 Metaschema for Workflow-Nets

Workflow-Nets are especially suited to describe and analyze workflows. Formally speaking (cf. [Aal00]), a Workflow-Net is a Petri-Net

- there exists one input (or source) place $i \in P$ with $\bullet i = \emptyset$, i.e. i is no postcondition of any transition,
- one output (or sink) place $o \in P$ with $o\bullet = \emptyset$, i.e. o is no precondition to any transition,
- every node $x \in P \cup R$ is on a path from i to o

Additionally, transitions can be augmented by triggers (time and event) describing additional conditions for firing specific transitions. We will use this additional construct within our example and, therefore, take into account within our meta schema.

To enable exchanging Workflow-Nets with GXL, all modeling constructs have to be defined in a GXL metaschema. This metaschema can be viewed as an extension of the metaschema of simple Petri-Nets already presented in [Wi00] by a means to associate typed triggers to transitions. Figure 12 shows a *GXL metaschema for Workflow-Nets*.

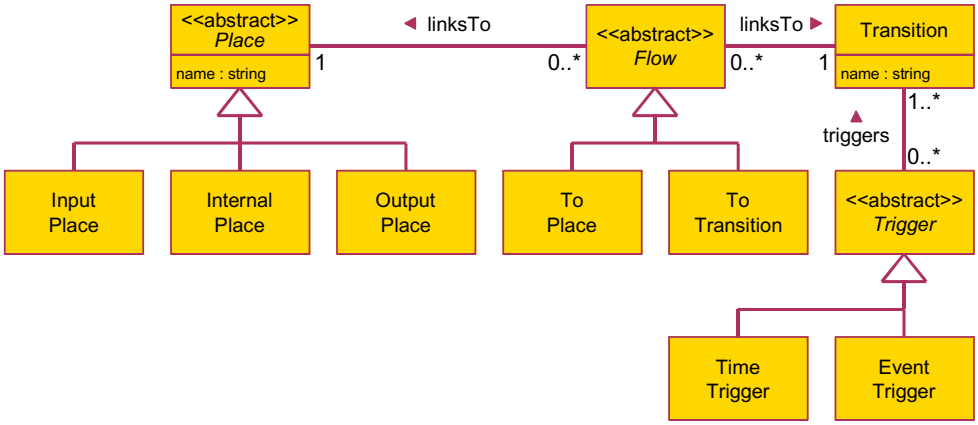


Figure 12: GXL meta schema for Workflow-Nets

Places and Transitions are connected by Flows. According to the definition of Workflow-Nets [Aal00], places are distinguished into InputPlaces, InternalPlaces, and OutputPlaces. Additional constraints ensure, that there exists only one InputPlace and only one OutputPlaces having no incoming or outgoing transitions. Transitions might be triggered by time (TimeTrigger) or by external events (EventTrigger). To show that GXL can deal with different meta-modeling styles, we used special subtypes of Flows to indicate if a flow links to a place (ToPlace) or to a transition (ToTransition). Corresponding places and transitions are connected by linksTo edges. A further constraint ensures that the resulting graph has to be connected.

Workflow-Nets can be exchanged with GXL, using the Workflow-Net metaschema from figure 12. The metaschema itself is exchanged by its suitable GXL graph (cf. section 2.2).

3.2.2 Exemplary Workflow-Net in GXL

Figure 13 shows an exemplary Workflow-Net taken from [Aal00]. The net describes the distribution and evaluation of questionnaires. Although this net is a simple Petri-Net except the extraordinary role of places i and o, two additional elements are added: a clock interpreted as a time trigger for transition time out and an envelope symbol representing an external trigger for transition process questionnaire. Their interpretation is as follows: transition time out fires immediately after a specified time is over, and transition process questionnaire fires immediately when an external event occurs and is recognized - in this example the arrival of an answered questionnaire.

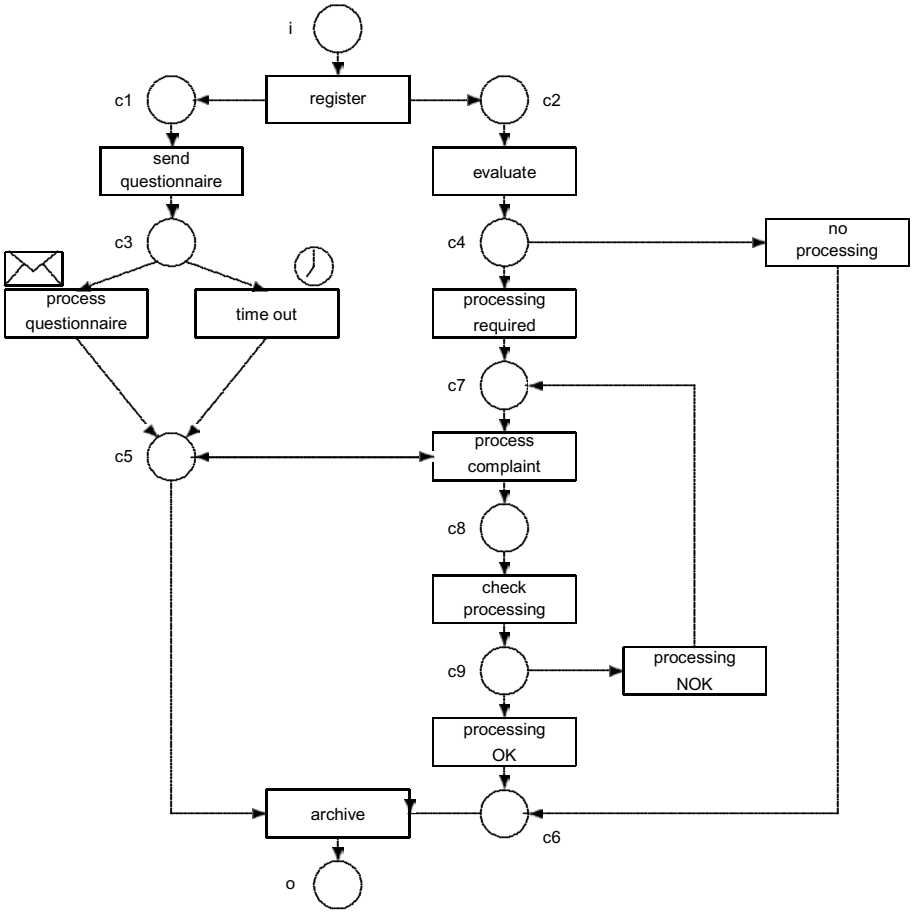


Figure 13: A WF-net for the processing of complaints from [Aal00]

Figure 14 shows the translation of the Workflow-Net into a graph matching the Workflow-Net metaschema from Figure 12. Input- and output places are modeled by InputPlace and OutputPlace nodes `c0` and `c10`. InternalPlace nodes represent all intermediate places, e. g. `c1:InternalPlace` describes a system state after registration. Transitions are modelled by Transition nodes, which also carry the transitions name: here the first transition is mapped to node `t1`. Place and Transition nodes are connected by ToTransition and ToPlace nodes. These nodes are associated by linksTo edges. Triggers controlling the invocation of transition *process questionaire* and *time out* are modeled by `x1:TimeTrigger` and `x2:EventTrigger` nodes, respectively. They are connected to their Transitions by triggers edges.

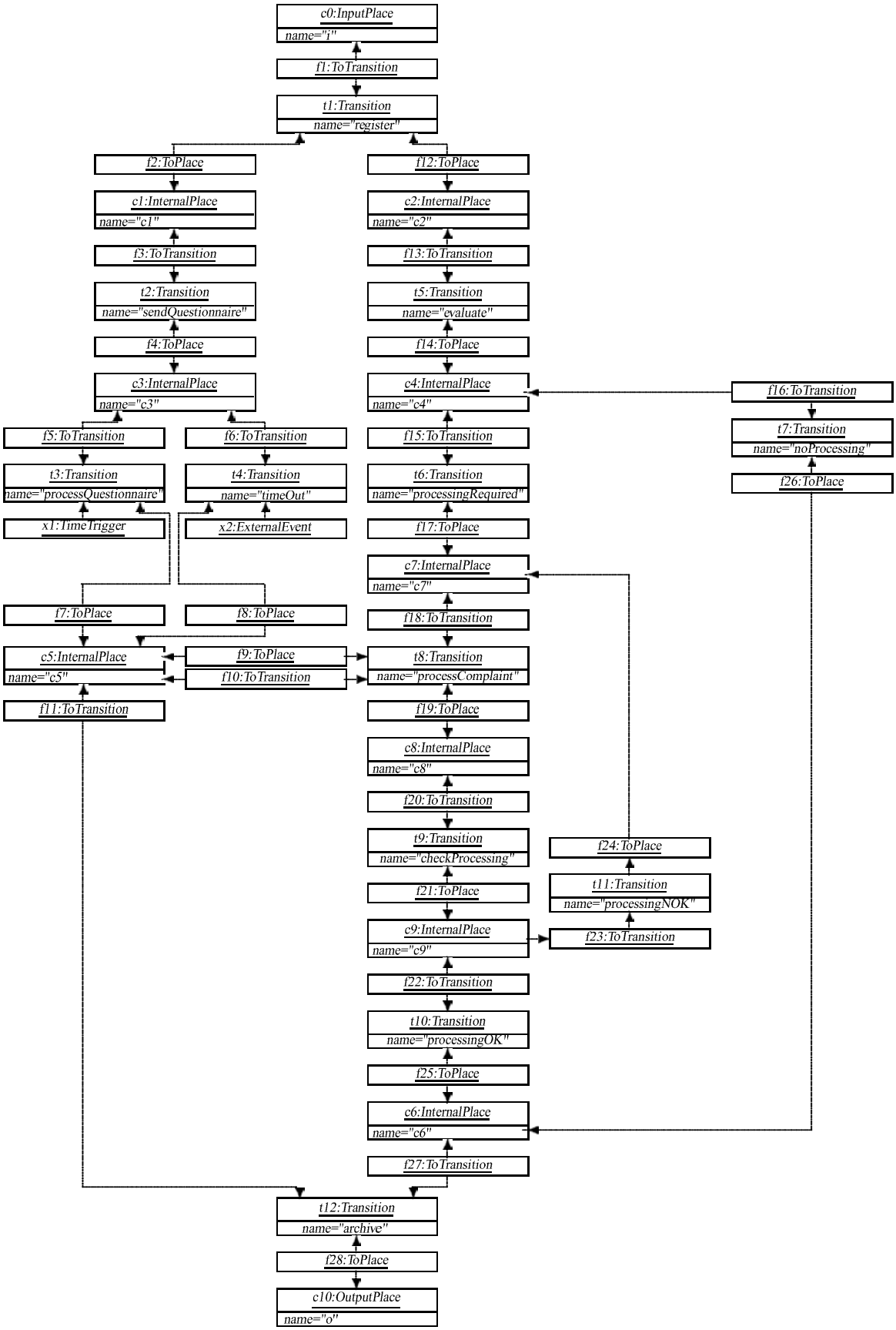


Figure 14: Representation of the exemplary Workflow-Net of Figure 13 as UML instance diagram

Finally, the graph depicted in Figure 14 has to be transfered into a GXL stream. Figure 15 shows a snippet of that document referring to the Workflow-Net metaschema (cf. Figure 12) stored in as GXL document in `metaPN.gxl`. Places, transitions, and triggers are exchanged by suitable <**node**> elements (lines 6-19). <**edge**> elements, referring the definition of linksTo and triggers-edges in `metaPN.gxl` describe the interrelation between places, transitions, and triggers.

```
1  <?xml version="1.0"?>
2  <!DOCTYPE gxl SYSTEM "http://www.gupro.de/GXL/gxl-1.0.dtd">
3  <gxl xmlns:xlink="http://www.w3.org/1999/xlink">
4  <graph id="examplePN" edgeids="true" edgemode="directed">
5    <type xlink:href="metaPN.gxl#pnSchema"/>
6    <node id="c0"><type xlink:href="pnSchema.gxl#InputPlace"/>
7      <attr name="name"><string>i</string></attr></node>
8  ...
9    <node id="c1"><type xlink:href="pnSchema.gxl#InternalPlace"/>
10     <attr name="name"><string>i</string></attr></node>
11 ...
12   <node id="t1"><type xlink:href="pnSchema.gxl#Transition"/>
13     <attr name="name"><string>define CO</string></attr></node>
14 ...
15   <node id="f1"><type xlink:href="pnSchema.gxl#ToTransition"/></node>
16 ...
17   <node id="f2"><type xlink:href="pnSchema.gxl#ToPlace"/></node>
18 ...
19   <node id="x1"><type xlink:href="pnSchema.gxl#TimeTrigger"/></node>
20 ...
21   <edge id="e1" from="f1" to="c0">
22     <type xlink:href="pnSchema.gxl#linksTo"/></edge>
23   <edge id="e2" from="f1" to="t1">
24     <type xlink:href="pnSchema.gxl#linksTo"/></edge>
25 ...
26   <edge id="e11" from="x1" to="t3">
27     <type xlink:href="pnSchema.gxl#triggers"/></edge>
28 ...
29 </graph>
30 </gxl>
```

Figure 15: GXL representation of the Workflow-Net graph in Figure 14 (extract)

4 Conclusion

GXL provides an adaptable, easy processable and widely distributed exchange format for graph-based data. Consequently, GXL is an eligible means to exchange models for all types of visual modeling languages. GXL follows a metamodel-based strategy to adapt GXL for exchanging particular models. Metaschemas for each modeling approach define graph structures, which carry appropriate models. In this paper we demonstrate how to tailor GXL to exchange business process models depicted as Event-driven Process Chains and Workflow-Nets. Analogously, GXL can be customized to exchange contol flow aspects of business process models represented in further Petri-Net based notations or process modeling languages like UML activity diagrams or flow charts. Exchanging data on the information-view or the organizational-structure-view of business process follows the same mechanism by using suitable metaschemas for e. g. class diagrams, entity-relationship diagrams or organization charts. Integrated GXL schemas provide a coherent interchange format covering all views of business processes.

There exist further approaches for exchanging control flow aspects of business process models. *EPML (Event-Driven Process Chains (EPC) Markup Language)* [EPML], [MN04] offers an XML-based interchange format for Event-driven Process Chains. An XML-based interchange language for various types of Petri-Nets is given by *PNML (Petri Net Markup Language)* [PNML] [BCH+03]. Both offer means to exchange business process models including layout information. Whereas these approaches provide sufficient support for exchanging only EPC or Petri-Nets, GXL is not restricted to one style of modeling languages.

A general interchange format is given by XMI (XML Meta Data Interchange) [OMG00]. This approach comes along with different document type definitions for different modeling approaches. Usually, these document definitions are rather complex and contain a vast number of (unnecessary) XML elements. Additionally, XMI requires different document types for schemas and instances. Depending on the intended usage, a schema has to be represented as an XML-instance of its schema or as a document type definition. GXL only requires *one* common and simple document type definition (using only 18 elements) for exchanging instance and schema data in the same way.

In contrast to EPML and PNML, GXL does not offer any support for exchanging layout information on graph based data per se. Here, GXL follows a strong separation of content and layout. GXL provides means for exchanging content information. Layout information can be stored for instance with GraphML (Graph Markup Language) [BEH+01], [Gra01] or SVG (Scalable Vector Graphics) [W3C03]. Thus, storing business process models with GXL-files enables independent calculation of layout with proper graph layout techniques. But, since GXL is generally adaptable by metaschemas, these metaschemas might also be defined to carry structures for exchanging layout information.

Summarizing, GXL offers a general means for exchanging graph-based data. The adaptability of GXL enables its usage to exchange business process models following different styles. Expanding GXL to a general base for exchanging business process models, requires to agree upon a set of widely accepted schemas for all commonly used business process models. These schemas might also found a base for defining transformations between different business process languages A catalog of candidates for those metaschemas for a large variety of modeling languages is given in [Wi00].

References

[Apache] The Apache XML Project. `http:http://xml.apache.org/`.

[Ba96] Baumgarten, B.: *Petri-Netze, Grundlagen und Anwendungen*. Heidelberg. 1996.

[BCH+03] Billington, J., Christensen, S., van Hee, K., Kindler, E., Kummer, O., Petrucci, L., Post, R., Stehno, C., and Weber, M.: The Petri Net Markup Language: Concepts, Technology, and Tools. In: *W. van der Aalst, E. Best (eds.): Applications and Theory of Petri Nets 2003: 24th International Conference, ICATPN 2003. Proceedings, LNCS 2679*. S. 483–505. 2003.

[Be76] Berge, C.: *Graphs and Hypergraphs*. volume 6. North-Holland. Amsterdam. 2. 1976.

[BEH+01] Brandes, U., Eiglsperger, M., Herman, I., Himsolt, M., and Marschall, M. S.: GraphML Progress Report, Structural Layer Proposal. In: *Graph Drawing 2001*. 2001.

[BH99] Bungert, W. and Heß, H.: Objektorientierte Geschäftsmodellierung. *Information Management*. 10(1):52–63. 1999.

[Bl04] Blaha, M.: Data Store Models are Different than Data Interchange Models (Workshop on Meta-Models and Schemas for Reverse Engineering) . *to appear in Electronic Notes on Theoretical Computer Science*. 2004.

[BRJ99] Booch, G., Rumbaugh, J., and Jacobson, I.: *The Unified Modeling Language User Guide*. Addison Wesley. Reading. 1999.

[Bu01] Busatto, G. An Abstract Model of Hierarchical Graphs and Hierarchical Graph Transformation. `http://www.informatik.uni-bremen.de/~giorgio/papers/phd-thesis.ps.gz`. 2001.

[CSMR02] *6th European Conference on Software Maintenance and Reengineering*. IEEE Computer Society. March 11 - 13 2002.

[EKM01] J. Ebert, K. Kontogiannis, J. Mylopoulos: Interoperability of Reverse Engineering Tools. `http://www.dagstuhl.de/DATA/Reports/01041/`. 2001.

[EPML] EPC Markup Languge. `http://wi.wu-wien.ac.at/Wer_sind_wir/mendling/EPML/`.

[EWD+96] Ebert, J., Winter, A., Dahm, P., Franzke, A., and Süttenbach, R.: Graph Based Modeling and Implementation with EER/GRAL . In: *[Th96]*. S. 163–178. 1996.

[FB02] Ferenc, R. and Beszédes, A.: Data Exchange with the Columbus Schema for C++. In: *[CSMR02]*. S. 59–66. 2002.

[FSH+01] Ferenc, R., Sim, S. E., Holt, R. C., Koschke, R., and Gyimòthy, T.: Towards a Standard Schema for C/C++. In: *[WCRE01]*. S. 49–58. 2001.

[Gra01] The GraphML Format. `http://www.graphdrawing.org/graphml/`. 2001.

[GTXL] Graph Transformation System Exchange Language. `http://tfs.cs.tu-berlin.de/projekte/gxl-gtxl.html`.

[GXLa] GXL: Graph Exchange Language. `http://www.gupro.de/GXL`.

[GXLb] GXL: Graph Exchange Language, Tools. `http://www.gupro.de/GXL/tools/tools.html`.

[HWS00] Holt, R. C., Winter, A., and Schürr, A.: GXL: Toward a Standard Exchange Format. In: *[WCRE00]*. S. 162–171. 2000.

[JBS97] Jablonski, S., Böhm, M., Schulze W.: *Workflow-Management, Entwicklung von Anwendungen und Systemen, Facetten einer neuen Technologie*. dpunkt. Heidelberg. 1997.

[JVW00] Janssens, G. K., Verelst, J., and Weyn, B.: Techniques for Modelling Workflows and Their Support of Reuse. In: van der Aalst, W., Desel, J., and Oberweis, A. (eds.), *Business Process Management (Models, Techniques, and Empirical Studies)*. LNCS 1806. Berlin. 2000. Springer.

[Ka03] Kaczmarek, A.: GXL Validator, Validierung von GXL-Dokumenten auf Instanz-, Schema, und Metaschema-Ebene. Studienarbeit. Universität Koblenz-Landau. 2003.

[KGW98] Koschke, R., Girard, J.-F., and Würthner, M.: An Intermediate Representation for Integrating Reverse Engineering Analyses. In: *[WCRE98]*. S. 241–250. 1998.

[KNS92] Keller, G., Nüttgens, M., and Scheer, A.-W.: Semantische Prozeßmodellierung auf der Grundlage "‘Ereignisgesteuerter Prozeßketten"’ (EPK). IWi-Heft Heft 89. Institut für Wirtschaftsinformatik. Saarbrücken. Januar 1992.

[LTP04] Lethbridge, T. C., Tichelaar, S., and Ploedereder, E.: The Dagstuhl Middle Metamodel, (Workshop on Meta-Models and Schemas for Reverse Engineering) . *to appear in Electronic Notes on Theoretical Computer Science*. 2004.

[MJL02] Mutzel, P., Jünger, M., and Leipert, S. (eds.): *Graph Drawing, 9th International Symposium, GD 2001 Vienna, Austria, September 23-26, 2001. Revised Papers*. *LNCS 2265*. Springer. Berlin. 2002.

[MN04] Mendling, J. and Nüttgens, M. EPC Markup Language and Refernce Models for XML Model Interchange. draft. 2004.

[Mü98] Müller, H. Criteria for Success, in Exchange Formats for Information Extracted from Computer Programs. http://plg2.math.uwaterloo.ca/ holt/sw.eng/exch.format/. 1998.

[NR02] Nüttgens, M. and Rump, F. J.: Syntax und Semantik Ereignisgesteuerter Prozessketten (EPK). In: *PROMISE 2002, Prozessorientierte Methoden und Werkzeuge für die Entwicklung von Informationssystemen, LNI P-21*. S. 64–77. 2002.

[OMG00] XML Meta Data Interchange (XMI) Specification. http://www.omg.org/technology/documents/formal/xmi.htm. November 2000.

[Pe62] Petri, C. A. Kommunikation mit Automaten. Schriften des Institutes für instrumentelle Mathematik, Bonn. 1962.

[PNML] Petri-Net Markup Language. http://www.informatik.hu-berlin.de/top/pnml/.

[Sc94a] Scheer, A.-W.: *Business Process Engineering - Reference Models for Industrial Enterprises, 2nd ed.* Springer-Verlag. Berlin. 1994.

[Sc94b] Scheer, A.-W.: *Wirtschaftsinformatik, Referenzmodelle für industrielle Geschäftsprozesse*. Springer. Berlin. 5. 1994.

[Sc00] Scheer, A.-W.: *ARIS - Business Process Modeling, 3rd ed.* Springer-Verlag. Berlin. 2000.

[ScY] Scotland Yard, Hunting Mr. X. Ravensburger.

[Si02a] Simon, C.: A logic of actions to specify and verify process requirements. In: *The Seventh Australien Workshop on Requirements Engineering (AWRE'2002)*. Melbourne, Australia. 2002.

[Si02b] Simon, C.: Verification in factory and office automation. In: *IEEE International Conference on Systems, Man and Cybernetics (SMC)*. Hammamet, Tunesien. 2002.

[St99] Staud, J.: *Geschäftsprozeßanalyse mit Ereignisgesteuerten Prozeßketten, Grundlage des Business Reengineering für SAP R/3 und andere Betriebswirtschaftliche Standardsoftware*. Springer. Berlin. 1999.

[Ta01] Taentzer, G.: Towards Common Exchange Formats for Graphs and Graph Transformation Systems. In: *Proceedings UNIGRA satellite workshop of ETAPS'01*. 2001.

[Th96] Thalheim, B. (ed.): *Conceptual Modeling — ER'96. LNCS 1157*. Springer. Berlin. 1996.

[Aal96] van der Aalst, W.: Structural Characterizations of Sound Workflow Nets. Computing Science Reports 96/23. Eindhoven University of Technology. 1996.

[Aal00] van der Aalst, W.: Workflow Verification: Finding Control-Flow Errors Using Petri-Net-Based Techniques. In: van der Aalst, W., Desel, J., and Oberweis, A. (eds.), *Business Process Management (Models, Techniques, and Empirical Studies)*. LNCS 1806. Berlin. 2000. Springer.

[ADK02] van der Aalst, W., Desel, J., and Kindler, E.: On the semantics of EPCs: A vicious circle. In: *M. Nüttgens, F. J. Rump (eds): EPK 2002, Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten.* S. 71–79. 2002.

[AH02] van der Aalst, W. and van Hee, K.: *Workflow Management - Models, Methods, and Systems*. MIT Press. Cambridge, Massachusetts. 2002.

[W3C00] Extensible Markup Language (XML) 1.0 (Second Edition). http://www.w3.org/TR/2000/REC-xml-20001006.pdf. October 2000.

[W3C01] XML Linking Language (XLink) Version 1.0, W3C Recommendation. http://www.w3.org/TR/2001/REC-xlink-20010627/. 27 June 2001.

[W3C03] Scalable Vector Graphics (SVG) 1.1 Specification. http://www.w3.org/TR/2003/REC-SVG11-20030114/. 14 January 2003.

[WCRE98] *5th Working Conference on Reverse Engineering*. IEEE Computer Society. 1998.

[WCRE00] *7th Working Conference on Reverse Engineering*. IEEE Computer Society. 2000.

[WCRE01] *8th Working Conference on Reverse Engineering*. IEEE Computer Society. 2001.

[WfM96] Terminology & Glossary. Technical Report WFMC-TC-1011. Workflow Management Coalition. Brussels. June 1996.

[Wi00] Winter, A.: *Referenz-Metaschemata für visuelle Modellierungssprachen*. Deutscher Universitätsverlag. Wiesbaden. 2000.

[Wi02] Winter, A.: Exchanging Graphs with GXL. In: *[MJL02]*. S. 485–500. 2002.

[WK98] Warmer, J. B. and Kleppe, A. G.: *The Object Constraint Language : Precise Modeling With UML*. Addison-Wesley. 1998.