Noname manuscript No.
(will be inserted by the editor)

Using GXL for Exchanging Business Process Models

Andreas Winter[1] **and Carlo Simon**[2]

[1] Institute for Software Technology, University Koblenz-Landau, D-56070 Koblenz, Universitätsstraße 1, e-mail: winter@uni-koblenz.de
[2] Institute for Management, University Koblenz-Landau, D-56070 Koblenz, Universitätsstraße 1, e-mail: simon@uni-koblenz.de

The date of receipt and acceptance will be inserted by the editor

Abstract The 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. This paper describes a novel application of GXL to the exchange of business process models. It is shown, how to customize GXL to exchange business process models depicted as Event-driven Process Chains and Workflow nets as examples for the control flow part of business process models. Each level of modeling is exemplarily demonstrated from the metaschemas down to instances of graphs.

Key words graphs – Event-driven Process Chains – Petri nets – exchange formats – interoperability

1 Introduction

Business Process Management typically deals with the description, optimization, and automization of an organization's business processes integrated into its information system. In the case of interorganizational processes, business partners have to exchange information about their individual processes and have to agree upon (partially) common processes. Thus, business process models must be exchanged between business partners on base of respectively used business process modeling tools. This is not only the technical task of mutually adapting software but it is also an organizational one affecting their strategic behavior. Hereby, the business processes of each of the partners specify behavior requested by potential cooperation partners. The verification whether relevant process behavior is still fulfilled in the cooperation is discussed for example in (Simon and Dehnert, 2004).

Consequently, there is a need for process specification exchange formats, i.e. business process models. Like models in software engineering, also business process models are typically *graph based*. Therefore, existing exchange formats in this field should be investigated concerning their applicability to business processes.

This paper introduces *GXL – Graph eXchange Language* as a general interchange format for business processes. GXL (Holt et al., 2000), (Winter, 2002) is an established format in software engineering which, however, has not been used in a business process context yet. GXL overcomes several limitations of current business process exchange formats as discussed in the following Section 2. The reminder of this paper introduces requirements for graph-based exchange formats (cf. Section 3). The foundation of GXL is presented in Section 4. Section 5.1 and Section 5.2 show how GXL is customized to exchange Event-driven Process Chains and Workflow nets. We finish the paper with some concluding remarks.

2 Related Work

There already exist approaches for exchanging business process models. In the following we will discuss languages used to exchange business process models depicted in certain notations, like EPML, PNML, and BPEL, as well as generic exchange notations like XMI.

EPML (Event-driven Process Chains Markup Language) (Mendling and Nuettgens, 2004), (EPML, 2004), offers an XML-based interchange format especially 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, 2004), (Billington et al., 2003).

These language specific notations are limited to provide sufficient support for exchanging only EPC or Petri nets. EPML and PNML, both offer means to exchange business process models including layout information. Due to separation of concerns, contemporary approaches to represent data, strongly separate content and layout information. Thus, an exchange format should also allow to separately store layout information.

Due to economical reasons, one of the currently most widely discussed languages for the specification of business processes is the Business Process Execution Language for Web Services (BPEL4WS or BPEL for short) (Andrews et al., 2003). As the name indicates, BPEL supports an exchange of process information in web service environments. The use of business protocols provides a means to formulate abstractions on internally complex business processes. BPEL does not intend to exchange business process models in general but only such aspects of businesses which are part of the BPEL specification. BPEL is also fixed to its underlying view on business processes. So, it does not cover the representation of several aspects of business process graphs like places in Workflow nets. Therefore, a general exchange format should be extendable and adaptable to cope with various modeling concepts used in various business process notations.

A general interchange format is given by XMI (XML meta data interchange) (OMG, 2000). To provide an adaptable exchange language XMI is based on meta-

modeling techniques. But, the used 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. A generally applicable and adaptable exchange format for business process models should provide a common notation independently from the current use of a model. Furthermore, exchange formats should also economize with document size.

The development of GXL aimed at simplicity and distribution in different domains of exchanging graph based data. Thus, GXL is not restricted to only one specific modeling language. The GXL format for exchanging both, instance and schema data, is kept very simple to avoid unneeded overhead and to keep GXL documents as short as possible. Therefore, GXL can be viewed as an alternative way to exchange business process models which is simple to use and generally applicable.

3 Exchanging models as graphs

Graphs are widely used for representing and analyzing structured data in various areas. They combine visual descriptiveness and clearness with mathematical foundation leading to efficient data-structures and -algorithms (e.g. (Tutte, 2001; Valiente, 2002)). 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. (Koschke et al., 1998)).

Adaptability: there exist several problem-related graph models such as trees, directed or undirected graphs, node and edge attributed graphs, node and edge typed graphs, hypergraphs, 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 data also requires to agree on the kind and structure of the data to be interchanged. Next to exchanging data, an exchange language has to provide an appropriate means to exchange structural information on data, e. g. the specification of graph structures.

Processability: the efficiency of exchange formats refers to the exchange process rather than the efficient usage of that data in certain applications (Blaha, 2004). Thus, each of the tasks 1) graph generation from stored data, 2) the actual exchange, and 3) the import must be conducted rapidly. For this, a set of tools supporting its usage must come along with a graph exchange language.

Distribution: standard exchange formats are only useful, if they are supported by a large number of tools. This demands that graph exchange languages support various components such as graph generation, analyzing (by applying graph algorithms), transformation, and visualization.

GXL was developed to fulfill these requirements:

Adaptability: GXL represents node/edge typed attributed ordered directed graphs (*TGraphs*) (Ebert et al., 1996) extended by *hierarchy* (Busatto, 2002) and *hypergraphs* (Berge, 1976). Thus, GXL covers most of usually used graph structures.

GXL supports both, exchanging graph structure (graph schema) and the actual graphs (instance graph). By mapping graph structures (depicted as class diagrams) to graphs in a standardized manner (Winter, 2002) GXL exchanges instance and schema information by using the same mechanisms.

Processability: GXL graphs and schemas are exchanged as *XML* (W3C, 2000) streams following the GXL document type definition (Winter, 2002) or the GXL XML Schema specification (GXL, 2004). The GXL DTD is defined plain and simple requiring 18 XML-elements only.

GXL gains profit from a variety of already existing XML tools like the Xerces parser for reading or Xalan (Apache, 2004) for manipulating GXL documents. Special GXL tools exist as for example for validating GXL documents (Kaczmarek, 2003). The only limitations of GXL are those resulting from the use of XML.

Distribution: GXL was proposed as standard exchange format in software reengineering at the Dagstuhl Seminar on *Interoperability of Reengineering Tools* in January 2001 (Dagstuhl Seminar 01041, 2001). It also builds the basis of the GTXL exchange language for Graph Transformation System (Taentzer, 2001) (GTXL, 2001). A still growing list of currently more than 40 tools e. g. for reengineering, graph transformation, or graph drawing supporting GXL can be found at (GXLTools, 2004).

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

GXL is also applicable 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 the following, we focus on languages modeling dynamic aspects of business processes. These languages include Event-driven Process Chains (Scheer, 1994b) and Petri net based approaches like Workflow nets (van der Aalst, 1996). Two examples which emphasize the specific modeling philosophies behind these modeling languages are used to demonstrate the applicability of GXL which allows to handle both types without any deficits.

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.

4 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, and, in a second (optional) part the graph structure is defined in a *graph schema*.

GXL originates in an exchange format for graph-based tools. It is applied to provide interoperability between various tools used in software-engineering. Here, we present a novel application of GXL for the exchange of visual business process models. To introduce the modeling concepts of GXL and its metamodeling facilities, the following section shows a plain example business process model and its GXL representation. A more complete version of this business process depicted as a Workflow net is given in section 5.2. Section 4.1 explains how business process models are mapped to graphs and how graph data is represented in GXL streams. The definition and exchange of graph schemas as special GXL graphs is presented in section 4.2.

4.1 Exchanging Graphs with GXL

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

Figure 1 depicts an exemplary plain business process modell describing the handling of questionnaires. The business process shows some activities (send questionnaire, process questionnaire, archive questionnaire, archive denied questionnaire) and some events (timeout, questionnaire received) which are connected by some arcs depicting the flow of control.

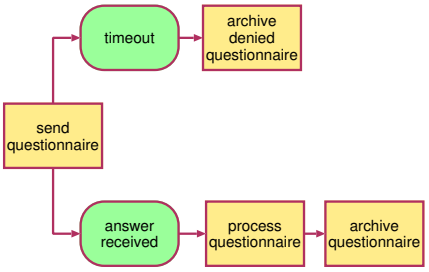

Fig. 1 a simple business process ...

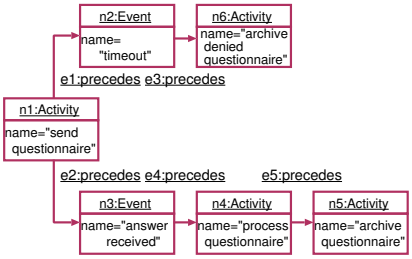

Fig. 2 ... represented as directed, typed, attributed graph

Models like this can easily be depicted as graphs. Each object (here activities and events) are represented by nodes and connections between them (here control

flow arcs) by edges. A graph representation of that map is the *directed, node- and edge typed, node attributed graph* depicted as object diagram in Figure 1.

The graph consists of two different kinds of nodes. Nodes of class Activity depict activities or subprocesses and Event-nodes emphasize states during processing. Both, Activity- and Event-nodes are characterized by their names. Edges of type precedes indicate the relative order of activities processing or events occurring.

GXL provides constructs for representing and exchanging such graphs. For this, constructs are required to represent nodes, edges, incidences, node- and edge-classes, and node-attributes.

Figure 3 depicts the complete GXL document representing the graph from Figure 1. 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 (Winter, 2002) and its XML-schema version is given at (GXL, 2004).

```
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="simpleBP" edgeids="true" edgemode="directed">
5    <type xlink:href="sBP.gxl#sBPSchema"/>
6    <node id="n1"><type xlink:href="sBP.gxl#Activity"/>
7      <attr name="name"><string>send questionnaire</string></attr></node>
8    <node id="n2"><type xlink:href="sBP.gxl#Event"/>
9      <attr name="name"><string>timeout</string></attr></node>
10   <node id="n3"><type xlink:href="sBP.gxl#Event"/>
11     <attr name="name"><string>answer received</string></attr></node>
12   <node id="n4"><type xlink:href="sBP.gxl#Activity"/>
13     <attr name="name"><string>process questionnaire</string></attr></node>
14   <node id="n5"><type xlink:href="sBP.gxl#Activity"/>
15     <attr name="name"><string>archive questionnaire</string></attr></node>
16   <node id="n6"><type xlink:href="sBP.gxl#Activity"/>
17     <attr name="name"><string>archive denied questionnaire</string></attr></node>
18   <edge id="e1" from="n1" to="n2"><type xlink:href="sBP.gxl#precedes"/></edge>
19   <edge id="e2" from="n1" to="n3"><type xlink:href="sBP.gxl#precedes"/></edge>
20   <edge id="e3" from="n2" to="n6"><type xlink:href="sBP.gxl#precedes"/></edge>
21   <edge id="e4" from="n3" to="n4"><type xlink:href="sBP.gxl#precedes"/></edge>
22   <edge id="e5" from="n4" to="n5"><type xlink:href="sBP.gxl#precedes"/></edge>
23 </graph>
24 </gxl>
```

Fig. 3 GXL representation of Figure 2

The body of the document is enclosed in a <**gxl**> element. 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 simpleBP, specifies that edges must have identifiers, and that the graph is interpreted as a directed graph. Nodes and edges are represented by <**node**> and <**edge**> elements, both located by their id-attribute. Incidences of edges are stored in from and to attributes within <**edge**> tags. These refer to the graph elements 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 (Warmer and Kleppe, 1998), 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 (W3C, 2001) <**type**> elements refer to a GXL-document defining the schema (here sBP.gxl) and the appropriate node or edge class. An extract of the GXL stream for that schema is shown in Figure 6. The relation between GXL graphs and their according schema can be checked within the GXL framework using the GXL validator (Kaczmarek, 2003).

In addition to the GXL features used in Figure 3, GXL provides the exchange of hypergraphs and hierarchical graphs. More information on exchanging various kinds of graphs can be found at (Winter, 2002).

4.2 Exchanging Graph Classes with GXL

Graphs like the one depicted in Figure 2 contain nodes representing activities and events. They are connected by edges representing logical or temporal relations between activities and events. Exchanging graphs this way requires knowledge about that structure. A graph schema defining the structure of graphs like the previous example is shown in Figure 4. The graph schema is depicted as an UML class diagram (Booch et al., 1999) where classes define node classes and associations define edge classes.

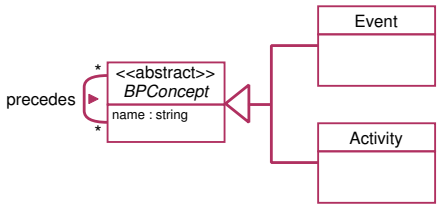

Fig. 4 Graph schema

The graph schema defines an abstract node class BPConcept and its specializations Event and Activity for modeling the relevant business process concepts event and activity. Nodes of class BPConcept are attributed by a string valued name attribute. All nodes may be connected by edges of edge class precedes.

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 Metaschema* (Winter, 2002). A graph representation of the schema in Figure 4 is depicted in Figure 5.

Node classes and edge classes are modeled by NodeClass-nodes and EdgeClass-nodes. Their name attribute determines the node and edge classes' name.

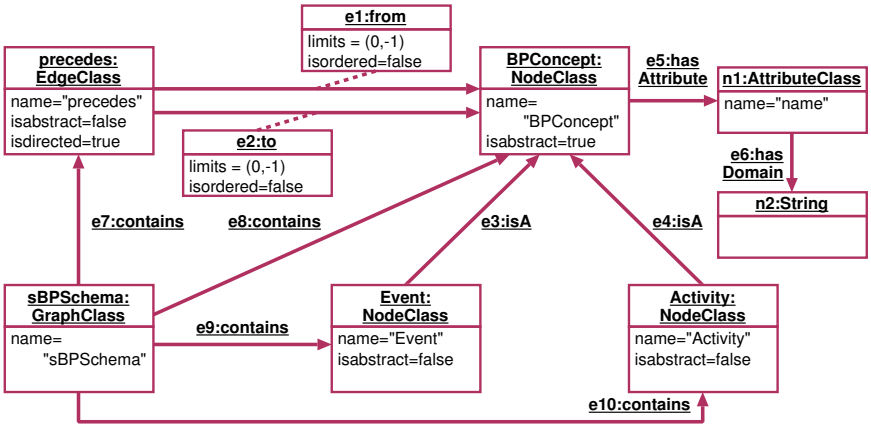

Fig. 5 Graph schema represented as Graph

For example, node class BPConcept is represented by the NodeClass node with OID `BPConcept` and edge class precedes is depicted by the EdgeClass node with OID `precedes`. Attribute isabstract indicates whether a class is treated as abstract (cf. abstract node class BPConcept) and isdirected=true labels directed edge classes. from and to-edges connect EdgeClasses with the incident NodeClasses. Their attributes contain multiplicities (limits) and indicate the treatment of incidences 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 (here: the String node). GXL provides generalization of node and edge classes by isA-edges; cf. edge $e3$ connecting `Event:NodeClass` and `BPConcept:NodeClass`. The complete graph class is represented by a GraphClass node which collects its node and edge classes by contains edges.

Graph class sBPSchema in Figure 5 contains node classes BPConcept, Event and Activity and edge class precedes. GXL instances matching that schema refer to these nodes as schema references, e. g. node `n1` in Figure 3 (line 6) refers to node `Activity` in the corresponding schema identifying `n1` as an activity.

Like all GXL graphs, the schema graph in Figure 5 can also be stored as an XML stream following the GXL definition. An extract of that GXL document is depicted in Figure 6. 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. node `BPConcept:NodeClass` refers to the definition of its type in the GXL-Metaschema (line 10). Analogously, lines 18-22 show the definition of edgeClass `precedes`. precedes edges connect BPConcept nodes. These incidences are modeled by edges `e1` and `e2` (lines 24-31). The specialization of BPConcept into Event is depicted by an appropriate isA edge (lines 33-34). Each GXL document defining a graph schema possesses at least one GraphClass node representing the graph class itself. Lines 6-8 denote the GraphClass node for the graph class. It is connected to all of its node and edge classes (e. g. lines 36-37).

```
<?xml version="1.0"?>
<!DOCTYPE gxl SYSTEM "http://www.gupro.de/GXL/gxl-1.0.dtd">
<gxl xmlns:xlink="http://www.w3.org/1999/xlink">
<graph id="sBP" edgeids="true" edgemode="directed">
  <type xlink:href="http://www.gupro.de/GXL/gxl-1.0.gxl#gxl-1.0"/>
  <node id="sBPSchema">
    <type xlink:href="http://www.gupro.de/GXL/gxl-1.0.gxl#GraphClass"/>
    <attr name="name"><string>sBPSchema</string></attr></node>
  <node id="BPConcept">
    <type xlink:href="http://www.gupro.de/GXL/gxl-1.0.gxl#NodeClass"/>
    <attr name="name"><string>BPConcept</string></attr>
    <attr name="isabstract"><bool>true</bool></attr></node>
  <node id="Event">
    <type xlink:href="http://www.gupro.de/GXL/gxl-1.0.gxl#NodeClass"/>
    <attr name="name"><string>Event</string></attr>
    <attr name="isabstract"><bool>false</bool></attr></node>
  ...
  <node id="precedes">
    <type xlink:href="http://www.gupro.de/GXL/gxl-1.0.gxl#EdgeClass"/>
    <attr name="name"><string>precedes</string></attr>
    <attr name="isabstract"><bool>false</bool></attr>
    <attr name="isdirected"><bool>true</bool></attr></node>
  ...
  <edge id="e1" from="precedes" to="BPConcept">
    <type xlink:href="http://www.gupro.de/GXL/gxl-1.0.gxl#from"/>
    <attr name="limits"><tup><int>0</int><int>-1</int></tup></attr>
    <attr name="isordered"><bool>false</bool></attr></edge>
  <edge id="e2" from="precedes" to="BPConcept">
    <type xlink:href="http://www.gupro.de/GXL/gxl-1.0.gxl#to"/>
    <attr name="limits"><tup><int>0</int><int>-1</int></tup></attr>
    <attr name="isordered"><bool>false</bool></attr></edge>

  <edge id="e3" from="Event" to="BPConcept">
    <type xlink:href="http://www.gupro.de/GXL/gxl-1.0.gxl#isA"/></edge>
  ...
  <edge id="e7" from="sBPSchema" to="precedes">
    <type xlink:href="http://www.gupro.de/GXL/gxl-1.0.gxl#contains"/></edge>
  ...
</graph>
</gxl>
```

Fig. 6 GXL representation of schema graph in Figure 5 (extract)

Since the GXL Metaschema is a schema, it is represented by a 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 (GXL, 2004).

4.3 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 document following the GXL specification.

Using GXL for exchanging graph data in a particular application domain requires to constrain the form of graphs, i. e. specifiying 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 (node, edge and hyperedge) 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. (Lethbridge et al., 2004) defines a GXL reference schema for language independent representation of source code data and (Ferenc et al., 2001; Ferenc and Beszédes, 2002) 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. We demonstrate how to customize GXL to exchange two different process models known from literature, which stress the different modeling philosophies behind these approaches. Customizing GXL to express Event-driven Process Chains and Workflow nets did not require to extend GXL. GXL's metamodel based adaptability provides using the same language for exchanging business process models in both notations.

5 Representation of Business Process Models

Business process models must provide the following views on organizations: control flow, information, and organizational structure (Scheer, 1994b), (Jablonski et al. , 1997, 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 by defining convenient metamodels. A collection of GXL compliant metamodels for various visual languages describing control flow, information, and organizational structures is given in (Winter, 2000).

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 (Janssens et al., 2000). Beside the descriptive nature of models and the ability to verify the models either with standard Petri net analyzing techniques or against special process specifications (Simon, 2002a; Simon, 2002b), their automatic execution within a Workflow Management System (WfMC, 1996) 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.

We have chosen Event-driven Process Chains (EPCs) introduced in (Keller et al., 1992) and Workflow nets (WF nets) introduced by van der Aalst (van der Aalst, 1996), (van der Aalst and van Hee, 2002) as exemplary notations for business process modeling to illustrate our approach. We define a metaschema for each of these notations and demonstrate the instantiation of these models by examples.

5.1 Exchanging Event-driven Process Chains with GXL

Event-driven Process Chains (EPCs) (Keller et al., 1992) are one central concept within the ARIS business process framework basing on stochastic networks and Petri nets (Scheer, 2000).

Events (depicted by hexagons) and functions (depicted by ovals) are the basic elements of EPCs which are connected by a flow relation and additional connections used to span complex process structures. Figure 7 shows the basic elements of EPCs as defined in (Scheer, 1994a).

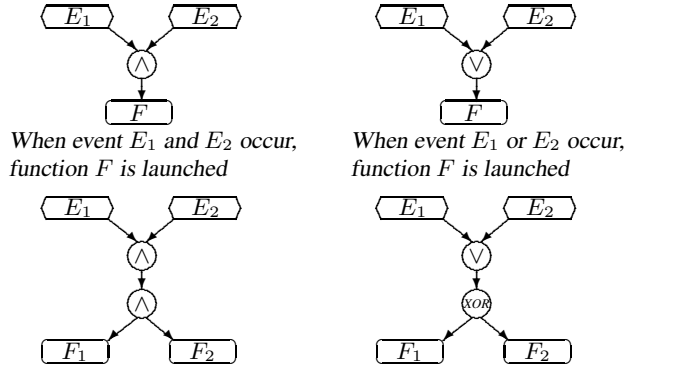

Fig. 7 Event relationships in EPC (Scheer, 1994a)

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. There exist various approaches to resolve this uncertainness by specifying the concrete syntax

(e. g. (Staud, 1999)) and semantics ((Nüttgens and Rump, 2002), (van der Aalst et al., 2002)) of EPC. In the following, we will not address that problem; we only specify the EPC-syntax as far as this is required to explain the idea of using GXL for exchanging EPC-based process models and assume an intuitive EPC semantics.

5.1.1 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 (Scheer, 2000) all modeling techniques used within the ARIS business process framework are introduced along their metaschemas. The EPC metaschema in (Scheer, 2000, 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 which have to be distiguished while exchanging business processes.

A useful metaschema 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 operators on *control flow*. Figure 8 shows the class diagram part of a GXL schema for EPC derivated from (Winter, 2000, p 119). 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.

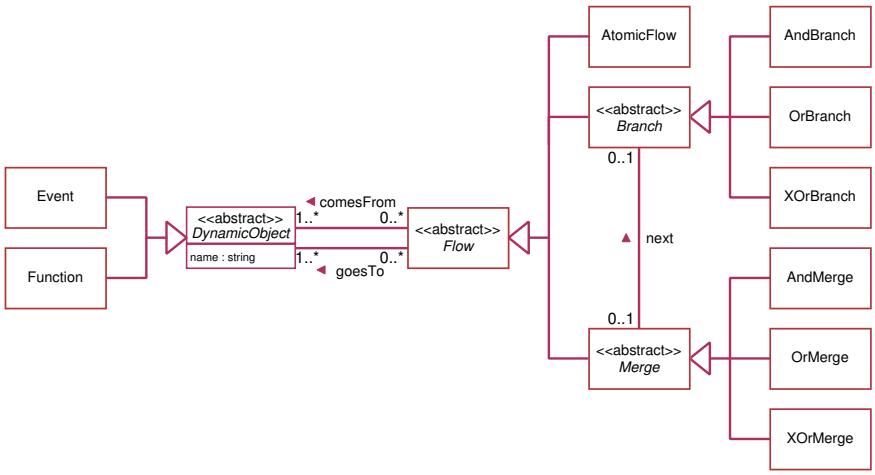

Fig. 8 GXL schema for event-driven process chains

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 7) 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, or 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 syntactic definition. Complete metaschemas for EPC with their class diagram and constraint part are given in (Winter, 2000, p 192). These schemas also include hierarchies of EPC and the embedding of EPC in multi-perspective modeling environments.

A metaschema like the one in Figure 8 controls the exchange of EPC business process models via GXL. According to the GXL metaschema the EPC schema is exchanged by an suitable GXL graph (cf. Section 4.2).

5.1.2 Exemplary EPC in GXL Exchanging an EPC with GXL is based on *translating* the EPC diagram into a GXL graph according the metaschema given in Figure 8. In the following, we will demonstate this with an example EPC modeling a business process for processing customer orders (Bungert and Heß, 1999, p 62) in Figure 9.

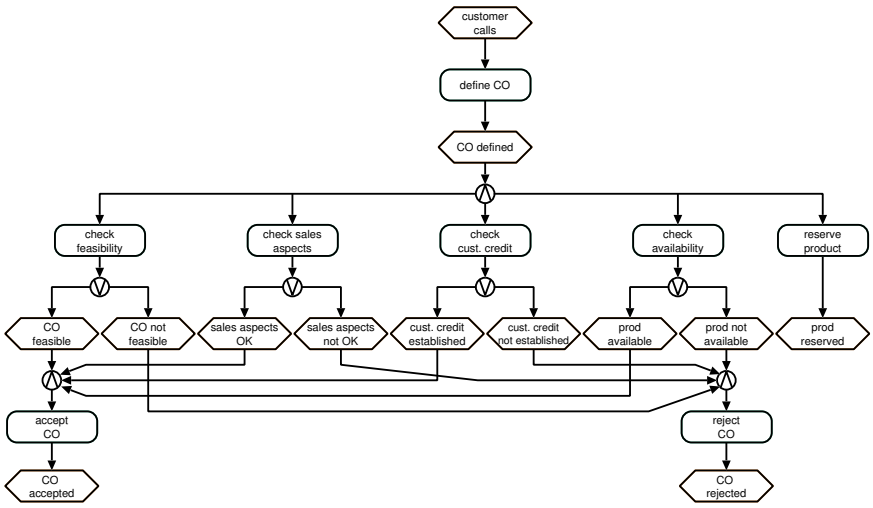

Fig. 9 An exemplary EPC "process customer order" (Bungert and Heß, 1999, p 62)

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 customer's order is either accepted or rejected.

Figure 10 shows an extract of the GXL graph representing the EPC depicted in Figure 9. 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*).

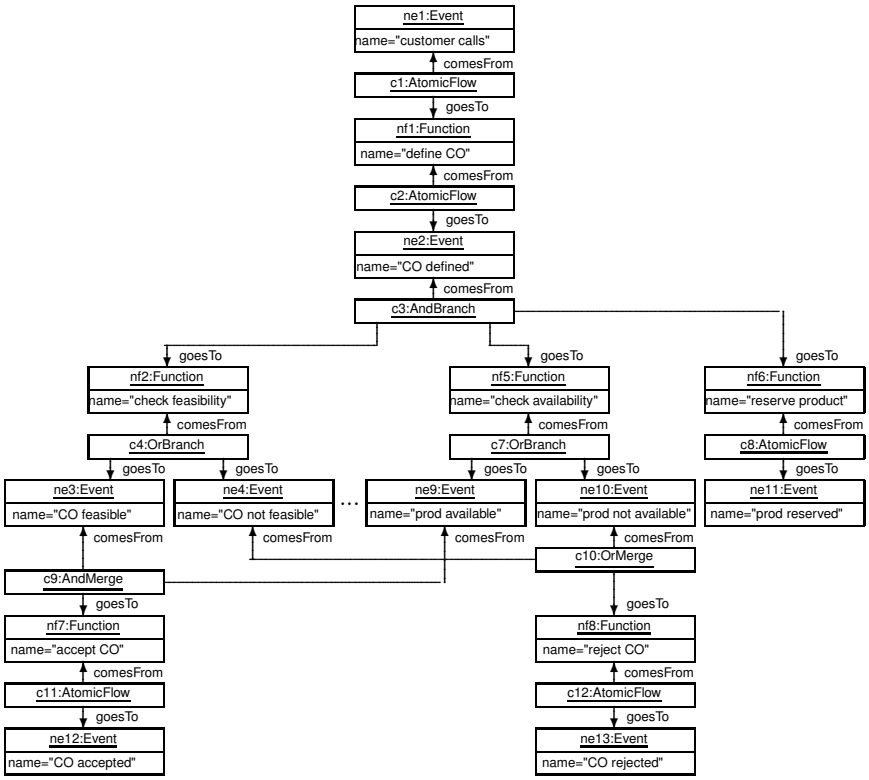

Fig. 10 Representation of the exemplary EPC of figure 9 as a UML instance diagram (extract)

On the contrary to the EPC diagram in Figure 9 where atomic flows of control are depicted as directed edges, the EPC metaschema 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* and 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*.

```
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="metaEPC.gxl#Event"/>
7      <attr name="name"><string>customer calls</string></attr></node>
8  ...
9    <node id="nf1"><type xlink:href="metaEPC.gxl#Function"/>
10     <attr name="name"><string>define CO</string></attr></node>
11 ...
12   <node id="c1"><type xlink:href="metaEPC.gxl#AtomicFlow"/></node>
13 ...
14   <node id="c3"><type xlink:href="metaEPC.gxl#AndBranch"/></node>
15 ...
16   <node id="c9"><type xlink:href="metaEPC.gxl#AndMerge"/></node>
17 ...
18   <edge id="e1" from="c1" to="ne1">
19     <type xlink:href="metaEPC.gxl#comesFrom"/></edge>
20 ...
21   <edge id="e17" from="c9" to="ne3">
22     <type xlink:href="metaEPC.gxl#comesFrom"/></edge>
23   <edge id="e18" from="c9" to="ne9">
24     <type xlink:href="metaEPC.gxl#comesFrom"/></edge>
25   <edge id="e22" from="c9" to="nf7">
26     <type xlink:href="metaEPC.gxl#goesTo"/></edge>
27 ...
28 </graph>
29 </gxl>
```

Fig. 11 GXL representation EPC graph in Figure 10 (extract)

Figure 11 shows the GXL stream representing an extract of the graph in Figure 10. This GXL document refers to the GXL representation of the EPC metaschema (cf. Figure 8) 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`).

5.2 Exchanging Workflow nets with GXL

Petri nets (Petri, 1962; Baumgarten, 1996) are a formally sound and well understood approach to describe and analyze 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 fundamental modeling constructs of those Place/Transition nets are shown in Figure 12.

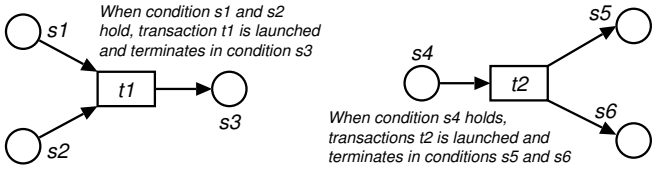

Fig. 12 Petri net modeling constructs

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 (van der Aalst, 1996) as an example.

5.2.1 Metaschema for Workflow nets Workflow nets are especially suited to describe and analyze workflows. Formally speaking (van der Aalst, 2000), a Workflow net is a Petri net if

- 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 $x \in P \cup T$ is on a path from i to o,

where P denotes the set of places and T denotes the set of transitions.

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 metaschema.

To enable exchanging Workflow nets with GXL, all modeling constructs have to be defined in the GXL metaschema. This metaschema can be viewed as an extension of the metaschema of simple Petri nets already presented in (Winter, 2000) by a means to associate typed triggers to transitions. Figure 13 shows a *GXL metaschema for Workflow nets*.

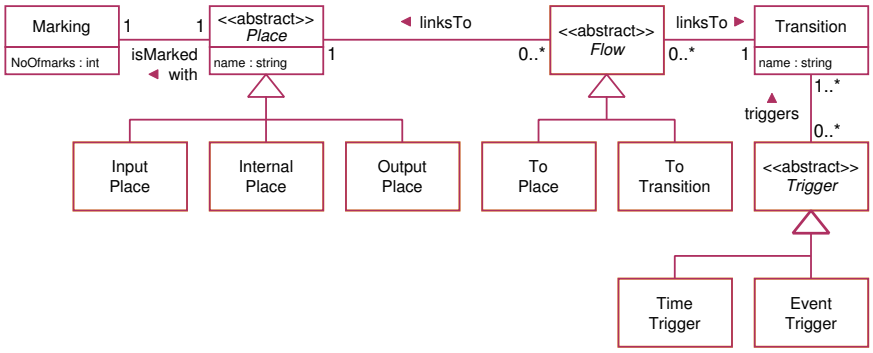

Fig. 13 GXL metaschema for workflow nets

Places and Transitions are connected by Flows. According to the definition of workflow nets (van der Aalst, 2000), places are distinguished into InputPlaces, InternalPlaces, and OutputPlaces. Additional constraints ensure that there exists exactly one InputPlace and exactly one OutputPlace having no incoming or outgoing transition respectively. Transitions might be triggered by time (TimeTrigger) or by external events (EventTrigger). Places are associated to Marking nodes to represent the (initial) marking. In this example, we only model the number of marks on a place. More sophisticated types of markings can be modeled analogeously. To show that GXL can deal with different meta-modeling styles, we use special subtypes of Flows in this example 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 in Figure 13. The metaschema itself is exchanged by its suitable GXL graph (cf. Section 4.2).

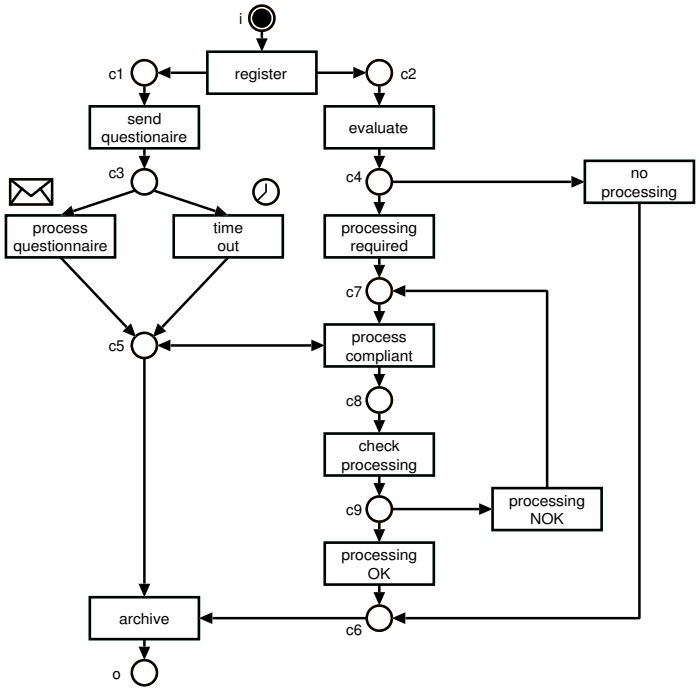

Fig. 14 An exemplary WF-net "processing complaint forms" (van der Aalst, 2000)

5.2.2 Exemplary Workflow net in GXL Figure 14 shows an exemplary Workflow net taken from (van der Aalst, 2000). The net describes the distribution and evaluation of questionnaires. Although this net is a simple Petri net except the ex-

traordinary role of places i (containing an initial marking) 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.

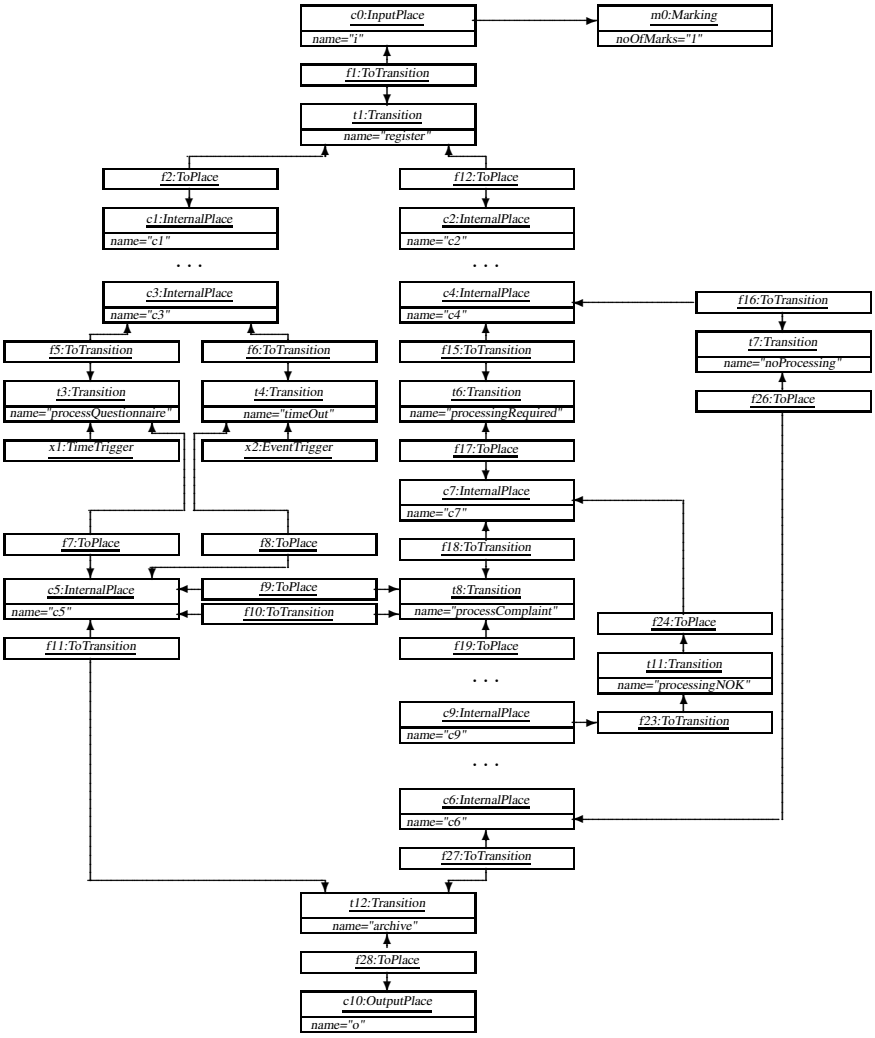

Fig. 15 Representation of the workflow-net of Figure 14 as UML instance diagram (extract)

Figure 15 shows (parts of) the translation of the Workflow net into a graph matching the Workflow net metaschema of Figure 13. 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 modeled by Transition nodes which also carry the transition's 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 questionnaire* and *time out* are modeled by `x1:TimeTrigger` and `x2:EventTrigger` nodes, respectively. They are connected to their transitions by triggers edges.

Finally, the graph depicted in Figure 15 has to be transfered into a GXL stream. Figure 16 shows a snippet of that document referring to the Workflow net metaschema (cf. Figure 13) stored as a GXL document in `metaPN.gxl`. Places, markings, transitions, and triggers are exchanged by suitable <**node**> elements (lines 6-22). <**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="metaPN.gxl#InputPlace"/>
7      <attr name="name"><string>i</string></attr></node>
8  ...
9    <node id="m0"><type xlink:href="metaPN.gxl#Marking"/>
10     <attr name="noOfMarks"><int>1</int></attr></node>
11 ...
12   <node id="c1"><type xlink:href="metaPN.gxl#InternalPlace"/>
13     <attr name="name"><string>c1</string></attr></node>
14 ...
15   <node id="t1"><type xlink:href="metaPN.gxl#Transition"/>
16     <attr name="name"><string>register</string></attr></node>
17 ...
18   <node id="f1"><type xlink:href="metaPN.gxl#ToTransition"/></node>
19 ...
20   <node id="f2"><type xlink:href="metaPN.gxl#ToPlace"/></node>
21 ...
22   <node id="x1"><type xlink:href="metaPN.gxl#TimeTrigger"/></node>
23 ...
24   <edge id="e1" from="f1" to="c0">
25     <type xlink:href="metaPN.gxl#linksTo"/></edge>
26   <edge id="e2" from="f1" to="t1">
27     <type xlink:href="metaPN.gxl#linksTo"/></edge>
28 ...
29   <edge id="e91" from="co" to="m0">
30     <type xlink:href="metaPN.gxl#isMarkedWith"/></edge>
31 ...
32   <edge id="e11" from="x1" to="t3">
33     <type xlink:href="metaPN.gxl#triggers"/></edge>
34 ...
35 </graph>
36 </gxl>
```

Fig. 16 GXL representation of the Workflow net graph in Figure 15 (extract)

6 Conclusion

As demonstrated in this paper, GXL is adaptable by metamodeling based customization, easyly processable by being based on XML, and widely distributed documented by its current use in various tools. 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 without extending the GXL language definition. Metaschemas for each modeling approach define graph structures which carry appropriate models. In this paper we demonstrated 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 processes 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.

Using GXL as exchange language for business process models overcomes deficites of allready proposed aproaches for exchanging business processes. Due to its metamodel-based adapatability GXL is not restricted to only one style of modeling language, and can be customized to exchange business process models in all visual notations. Furthermore, GXL follows a strong separation of content and layout. GXL provides means for exchanging content information only. Layout information can be stored for instance with GraphML (Graph Markup Language) (Brandes et al., 2002), (GraphML, 2001) or SVG (Scalable Vector Graphics) (W3C, 2003). 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 taylored to carry structures for exchanging layout information. GXL also requires only *one* common and simple document type definition (using only 18 elements) for exchanging instance and schema data in the same way. Thus, the language design is kept rather simple. Due to its foundation in XML, GXL is a verbose notation per se, but it requires significantly less space than appropriate XMI-based notations.

Summarizing, GXL offers a general means for exchanging graph-based data. The adaptability of GXL enables its usage to exchanging 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 (Winter, 2000).

References

Andrews, T, Curbera, F., Dholakia, H., Goland, Y. Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I, and Weerawarana, S. (2003). Business Process Execution Language for Web Services - Version 1.1. http://www6.software.ibm.com/software/developer/library/ws-bpel.pdf/ (29.12.2004).

Apache (2004). The Apache XML Project. http://xml.apache.org/ (29.12.2004).

Baumgarten, B. (1996). *Petri-Netze, Grundlagen und Anwendungen*. 2nd edn. Spektrum, Heidelberg.

Berge, C. (1976). *Graphs and Hypergraphs*, *North-Holland Mathematical Library*. 2nd edn. North-Holland, Amsterdam.

Billington, J., Christensen, S., van Hee, K., Kindler, E., Kummer, O., Petrucci, L., Post, R., Stehno, C., and Weber, M. (2003). 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, Eindhoven, The Netherlands, June 23-27, 2003. Proceedings, LNCS 2679*, pp 483–505.

Blaha, M. (2004). Data Store Models are Different than Data Interchange Models. *ENTCS*, vol. 94, pp 51–58.

Booch, G., Rumbaugh, J., and Jacobson, I. (1999). *The Unified Modeling Language User Guide*. Addison Wesley, Reading.

Brandes, U., Eiglsperger, M., Herman, I., Himsolt, M., and Marschall, M. S. (2001). GraphML Progress Report, Structural Layer Proposal. In 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. pp 501–512

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

Busatto, G. (2002). An Abstract Model of Hierarchical Graphs and Hierarchical Graph Transformation. Universität Paderborn, http://ubdata.uni-paderborn.de/ediss/17/2002/busatto/ (29.12.2004).

Dagstuhl Seminar 01041 (2001). J. Ebert, K. Kontogiannis, J. Mylopoulos: Interoperability of Reverse Engineering Tools. http://www.dagstuhl.de/DATA/Reports/01041/ (29.12.2004).

Ebert, J., Winter, A., Dahm, P., Franzke, A., and Süttenbach, R. (1996). Graph Based Modeling and Implementation with EER/GRAL . In Thalheim, B., ed. *Conceptual Modeling — ER'96, LNCS 1157*. Springer, Berlin, pp 163–178.

EPML (2004). EPC Markup Languge. http://wi.wu-wien.ac.at/Wer_sind_wir/mendling/EPML/ (29.12.2004).

Ferenc, R. and Beszédes, A. (2002). Data Exchange with the Columbus Schema for C++. In *6th European Conference on Software Maintenance and Reengineering, (CSMR 2002)*. IEEE Computer Society, Los Alamitos, pp 59–66.

Ferenc, R., Sim, S. E., Holt, R. C., Koschke, R., and Gyimòthy, T. (2001). Towards a Standard Schema for C/C++. In *8th Working Conference on Reverse Engineering (WCRE 2001)*. IEEE Computer Society, Los Alamitos, pp 49–58.

GraphML (2001). The GraphML File Format. http://www.graphdrawing.org/graphml (29.12.2004).

GTXL (2001). Graph Transformation System Exchange Language. http://tfs.cs.tu-berlin.de/projekte/gxl-gtxl.html (29.12.2004).

GXL (2004). GXL: Graph Exchange Language. http://www.gupro.de/GXL (29.12.2004).

GXLTools (2004). GXL: Graph Exchange Language. `http://www.gupro.de/GXL/tools/tools.html` (29.12.2004).

Holt, R. C., Winter, A., and Schürr, A. (2000). GXL: Toward a Standard Exchange Format. In *7th Working Conference on Reverse Engineering (WCRE 2000)*. IEEE Computer Society, Los Alamitos, pp 162–171.

Jablonski, S., Böhm, M., and Schulze, W., (eds) (1997). *Workflow-Management, Entwicklung von Anwendungen und Systemen, Facetten einer neuen Technologie*. dpunkt, Heidelberg.

Janssens, G. K., Verelst, J., and Weyn, B. (2000). 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*, Springer, Berlin.

Kaczmarek, A. (2003). *GXL Validator, Validierung von GXL-Dokumenten auf Instanz-, Schema und Metaschema-Ebene*. (Download: `http://www.uni-koblenz.de/FB4/Contrib/GUPRO/Site/Downloads/index_html?project=gxl` (28.12.2004), Universität Koblenz-Landau, Koblenz.

Keller, G., Nüttgens, M., and Scheer, A.-W. (1992). Semantische Prozeßmodellierung auf der Grundlage "'Ereignisgesteuerter Prozeßketten"' (EPK). *IWi-Heft* Heft 89, Institut für Wirtschaftsinformatik, Saarbrücken.

Koschke, R., Girard, J.-F., and Würthner, M. (1998). An Intermediate Representation for Integrating Reverse Engineering Analyses. In *5th Working Conference on Reverse Engineering (WCRE 1998)*. IEEE Computer Society, Los Alamitos, pp 241–250.

Lethbridge, T. C., Tichelaar, S., and Ploedereder, E. (2004). The Dagstuhl Middle Metamodel, A Schema for Reverse Engineering. *ENTCS*, vol. 94, pp 7–18.

Mendling, J. and Nüttgens, M. (2004). *EPC Markup Language and Refernce Models for XML Model Interchange*, draft.

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

OMG (2000). XML Meta Data Interchange (XMI) Specification. `http://www.omg.org/technology/documents/formal/xmi.htm` (29.12.2004).

Petri, C. A. (1962). *Kommunikation mit Automaten*. Schriften des Institutes für instrumentelle Mathematik, Bonn.

PNML (2004). *Petri-Net Markup Language*. `http://www.informatik.hu-berlin.de/top/pnml/about.html` (29.12.2004).

Scheer, A.-W. (1994a). *Business Process Engineering - Reference Models for Industrial Enterprises.* 2nd edn. Springer-Verlag, Berlin.

Scheer, A.-W. (1994b). *Wirtschaftsinformatik, Referenzmodelle für industrielle Geschäftsprozesse*. 5th edn. Springer, Berlin.

Scheer, A.-W. (2000). *ARIS - Business Process Modeling.* 3rd edn. Springer-Verlag, Berlin.

Simon, C. (2002a). A Logic of Actions to Specify and Verify Process Requirements. In *The Seventh Australien Workshop on Requirements Engineering (AWRE'2002)*, Melbourne, Australia.

Simon, C. (2002b). Verification in Factory and Office Automation. In *IEEE International Conference on Systems, Man and Cybernetics (SMC)*, Hammamet, Tunesia.

Simon, C. and Dehnert, J. (2004). From Business Process Fragments to Workflow Definitions. In *Feltz F., A. Oberweis and B. Otjacques, EMISA 2004 - Informationssysteme im E-Business und E-Government*, Lecture Notes in Informatics P-56, Luxembourg, pp 95–106.

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

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

Tutte, W. T. (2001). *Graph Theory*. Cambridge University Press.

Valiente, G. (2002). *Algorithms on Trees and Graphs*. Springer, Berlin.

van der Aalst, W. (1996). Structural Characterizations of Sound Workflow Nets. *Computing Science Reports*, 96/23, Eindhoven University of Technology.

van der Aalst, W. (2000). 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*. Springer, Berlin.

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

van der Aalst, W. and van Hee, K. (2002). *Workflow Management - Models, Methods, and Systems*. MIT Press, Cambridge, Massachusetts.

W3C (2000). Extensible Markup Language (XML) 1.0. `http://www.w3.org/TR/2000/REC-xml-20001006.pdf` (29.12.2004).

W3C (2001). XML Linking Language (XLink) Version 1.0, W3C Recommendation. `http://www.w3.org/ TR/2001/REC-xlink-20010627/` (29.12.2004).

W3C (2003). Scalable Vector Graphics (SVG) 1.1 Specification. `http://www.w3.org/TR/2003/REC-SVG11-20030114/` (29.12.2004).

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

WfMC (1996). Terminology & Glossary. Technical Report WFMC-TC-1011, Workflow Management Coalition, Brussels.

Winter, A. (2000). *Referenz-Metaschemata für visuelle Modellierungssprachen*. Deutscher Universitätsverlag, Wiesbaden.

Winter, A. (2002). Exchanging Graphs with GXL. In 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. pp 485–500.