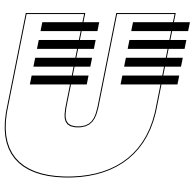

UNIVERSITÄT
KOBLENZ · LANDAU

Exchanging Business Process Models with GXL

Andreas Winter, Carlo Simon

1/2004

Fachberichte
INFORMATIK

Universität Koblenz-Landau
Institut für Informatik, Universitätsstr. 1, D-56070 Koblenz

E-mail: researchreports@uni-koblenz.de,
WWW: http://www.uni-koblenz.de/fb4/

Exchanging Business Process Models with GXL

Andreas Winter
Institute for Software Technology
University of Koblenz
D-56016 Koblenz
Universitätsstraße 1
www.uni-koblenz.de/~winter/
mailto:winter@uni-koblenz.de

Carlo Simon
Institute for Management
University of Koblenz
D-56016 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. Examples are used to demonstrate each level of modeling 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 sublanguage* [W3C00]. Graphs and schemas are exchanged by XML streams following the GXL document type definition or the GXL XML Schema specification [Wi02]. 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 sublanguages, 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 [apa]. 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 [Dag01]. GXL also defines the foundation of the GTXL exchange language for Graph transformation system [Ta01] [GTX]. 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 are expressed. Business processes are usually depicted using graphical languages focussing on dynamic aspects of systems. These languages include Event-driven Process Chains [Sc94b] and Petri-Net based approaches like Workflow-Nets [va96]. For the exchange of such 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 [sco]. 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

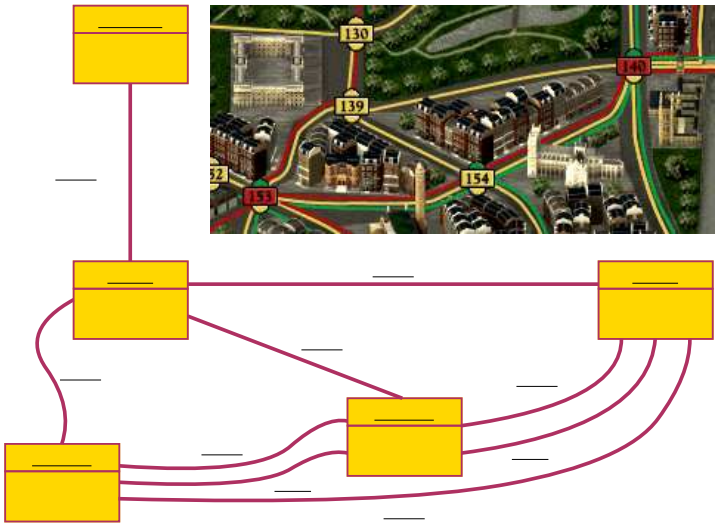

Figure 1: undirected typed, attributed graph

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 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].

The body of the document is enclosed in <**gxl**> tags. Line 3 includes the xlink-namespace, which is required for referring to external XML documents. Graphs are enclosed in <**graph**> tags. 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

```
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="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

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.

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.

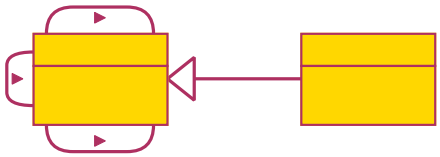

Figure 3: Graph schema

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 additionally 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.

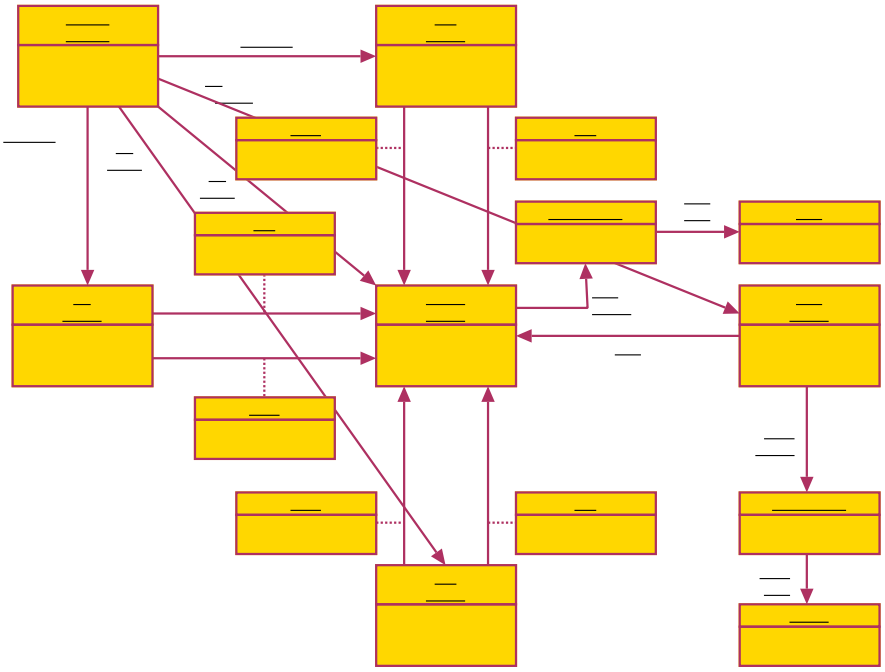

Figure 4: Graph schema represented as Graph

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.

Like all GXL graphs, this graph can be stored as an XML-stream following the GXL document type 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 DTD.

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

```
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)

- 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 modeling is one major application of Petri-Nets. 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 against a given specification [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 [va96, vv02] 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].

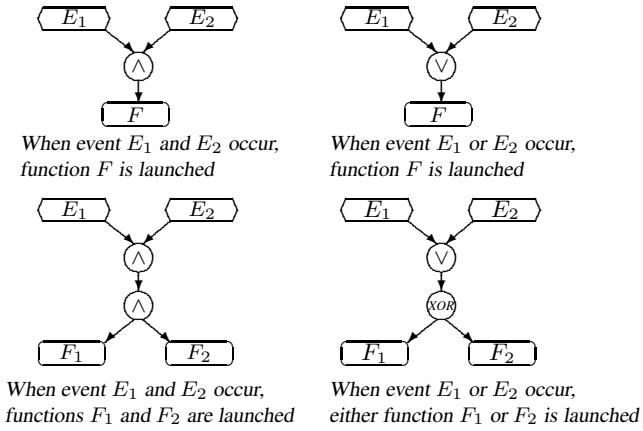

Figure 6: Event relationships in EPC from [Sc94a]

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], [vDK02]) 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.

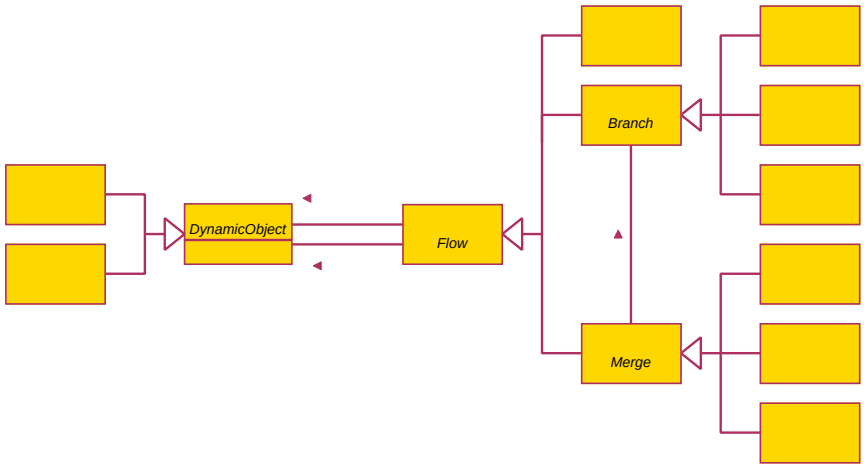

Figure 7: GXL schema for Event-driven Process Chains

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.

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.

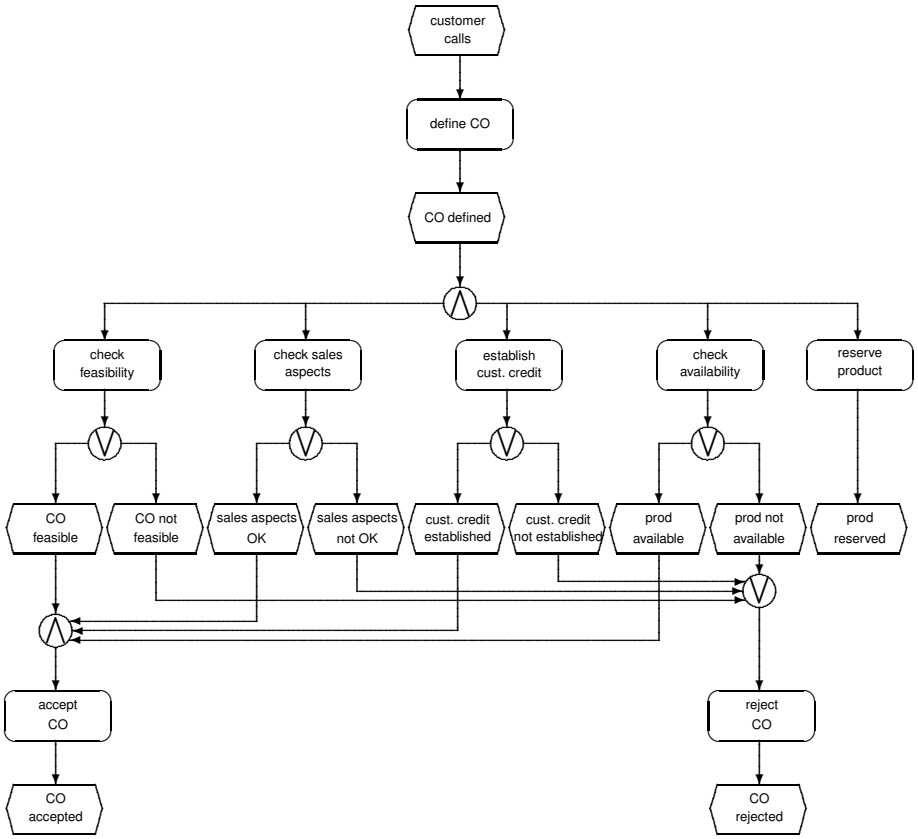

Figure 8: An exemplary EPC (from [BH99, p. 62])

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*).

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)

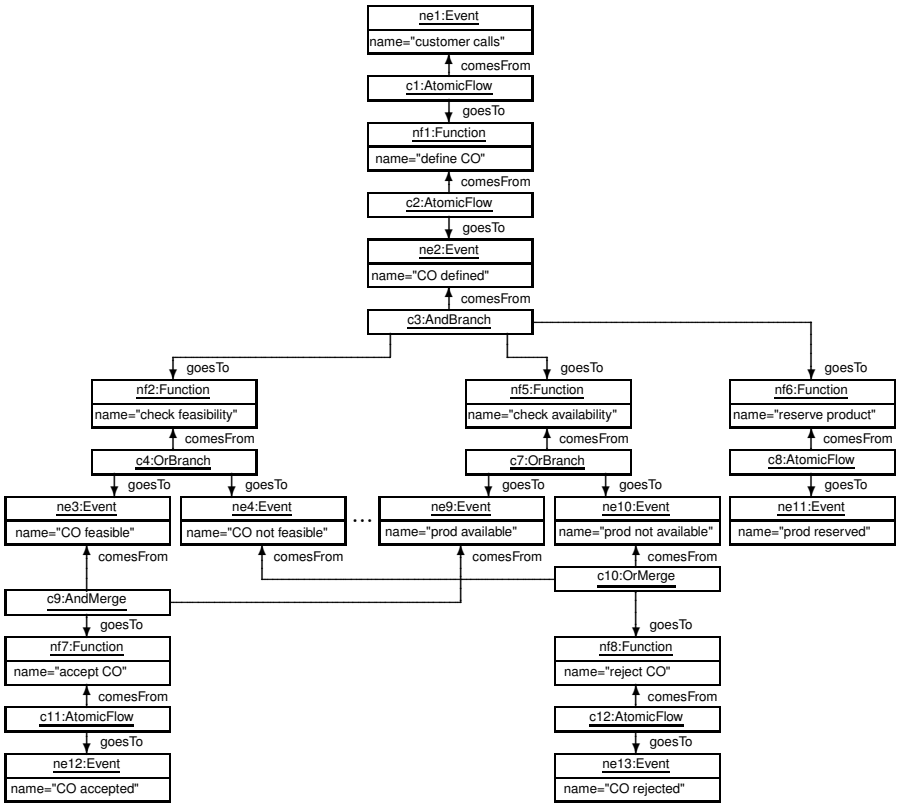

Figure 9: Representation of the exemplary EPC of figure 8 as a UML instance diagram

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).

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.

```
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)

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 [va96] as an example.

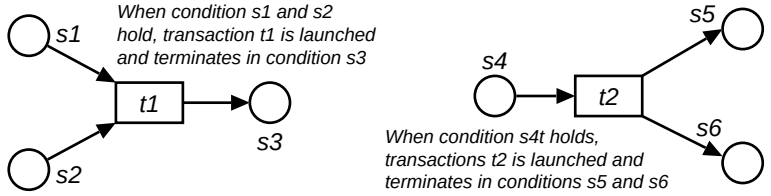

Figure 11: Petri-Net modeling constructs

3.2.1 Metaschema for Workflow-Nets

Workflow-Nets are especially suited to describe and analyze workflows. Formally speaking (cf. [va00]), 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*.

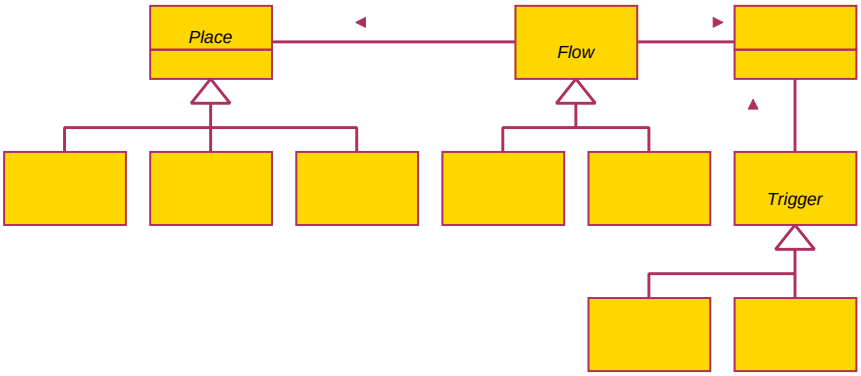

Figure 12: GXL meta schema for Workflow-Nets

Places and Transitions are connected by Flows. According to the definition of Workflow-Nets [va00], 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 [va00]. 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.

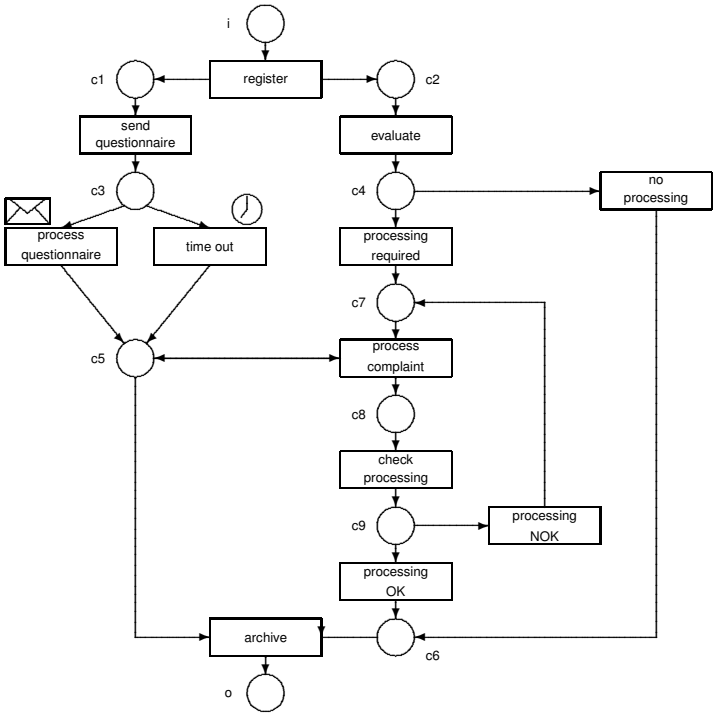

Figure 13: A WF-net for the processing of complaints from [va00]

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.

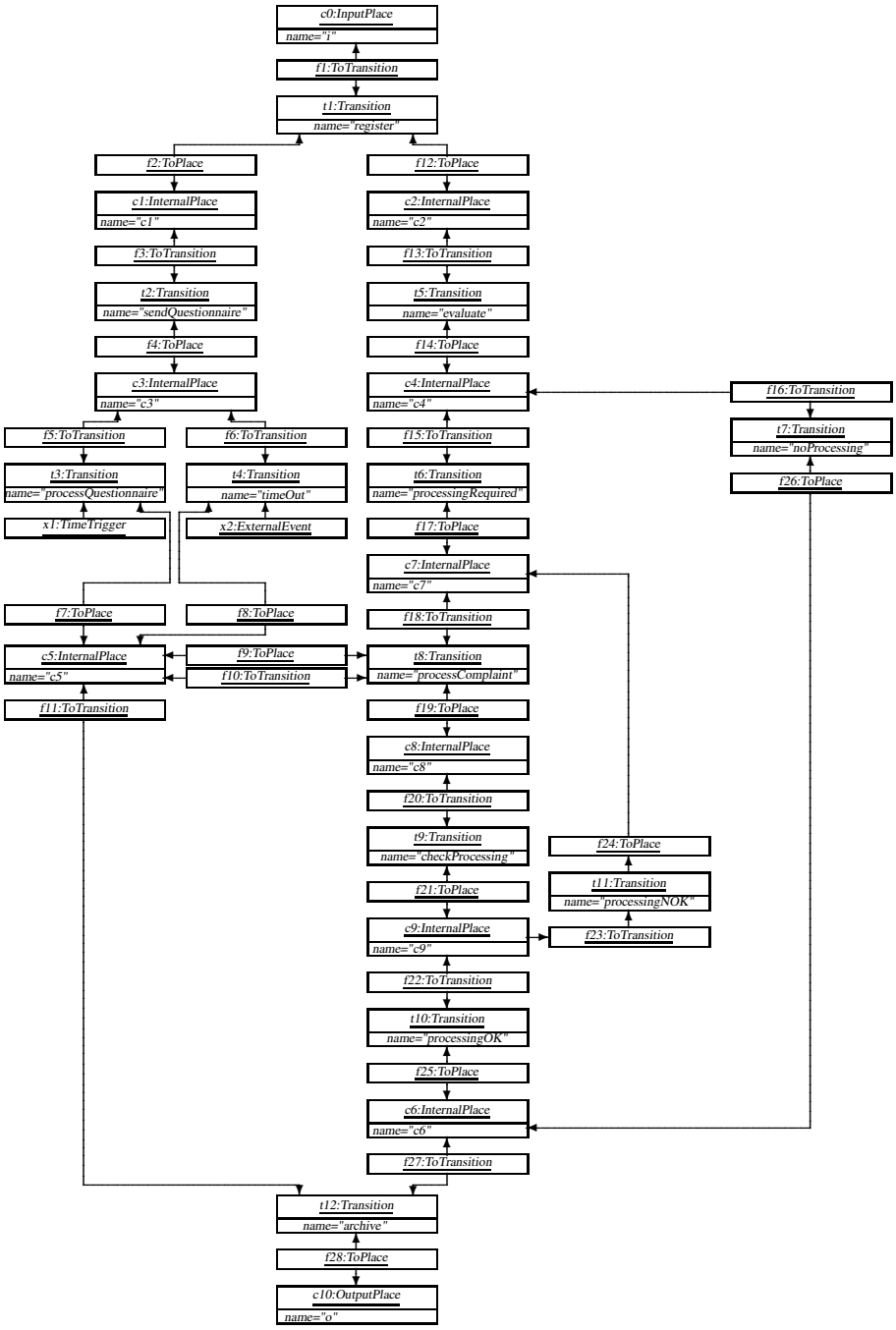

Figure 14: Representation of the exemplary Workflow-Net of Figure 13 as UML instance diagram

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 business process models represented in further Petri-Net based notations or process modeling languages like UML activity diagrams or flow charts.

There exist further approaches for exchanging business process models. EPML (Event-Driven Process Chains (EPC) Markup Language) [EPM], [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) [PNM] [BCv+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 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

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

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

[BCv+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.

[CSM02] *6th European Conference on Software Maintenance and Reengineering*. IEEE Computer Society. March 11 - 13 2002.

[Dag01] J. Ebert, K. Kontogiannis, J. Mylopoulos: Interoperability of Reverse Engineering Tools. `http://www.dagstuhl.de/DATA/Reports/01041/`. 2001.

[EPM] 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: *[CSM02]*. 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: *[WCR01]*. S. 49–58. 2001.

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

[GTX] 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. `http://www.gupro.de/GXL/tools/tools.html`.

[HWS00] Holt, R. C., Winter, A., and Schürr, A.: GXL: Toward a Standard Exchange Format. In: *[WCR00]*. S. 162–171. 2000.

[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)*. Number 1806 in Lecture Notes in Computer Science. Berlin. 2000. Springer.

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

[KGW98] Koschke, R., Girard, J.-F., and Würthner, M.: An Intermediate Representation for Integrating Reverse Engineering Analyses. In: *[WCR98]*. 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.

[PNM] 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.

[sco] 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.

[va96] van der Aalst, W.: Structural Characterizations of Sound Workflow Nets. Computing Science Reports 96/23. Eindhoven University of Technology. 1996.

[va00] 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)*. Number 1806 in Lecture Notes in Computer Science. Berlin. 2000. Springer.

[vDK02] 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.

[vv02] 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.

[WCR98] *5th Working Conference on Reverse Engineering*. IEEE Computer Society. 1998.

[WCR00] *7th Working Conference on Reverse Engineering*. IEEE Computer Society. 2000.

[WCR01] *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.

Available Research Reports (since 1999):

2004

1/2004 *Andreas Winter, Carlo Simon.* Exchanging Business Process Models with GXL.

2003

18/2003 *Kurt Lautenbach.* Duality of Marked Place/Transition Nets.

17/2003 *Frieder Stolzenburg, Jan Murray, Karsten Sturm.* Multiagent Matching Algorithms With and Without Coach.

16/2003 *Peter Baumgartner, Paul A. Cairns, Michael Kohlhase, Erica Melis (Eds.).* Knowledge Representation and Automated Reasoning for E-Learning Systems.

15/2003 *Peter Baumgartner, Ulrich Furbach, Margret Gross-Hardt, Thomas Kleemann, Christoph Wernhard.* KRHyper Inside — Model Based Deduction in Applications.

14/2003 *Christoph Wernhard.* System Description: KRHyper.

13/2003 *Peter Baumgartner, Ulrich Furbach, Margret Gross-Hardt, Alex Sinner.* 'Living Book' :- 'Deduction', 'Slicing', 'Interaction'..

12/2003 *Heni Ben Amor, Oliver Obst, Jan Murray.* Fast, Neat and Under Control: Inverse Steering Behaviors for Physical Autonomous Agents.

11/2003 *Gerd Beuster, Thomas Kleemann, Bernd Thomas.* MIA - A Multi-Agent Location Based Information Systems for Mobile Users in 3G Networks.

10/2003 *Gerd Beuster, Ulrich Furbach, Margret Gro-Hardt, Bernd Thomas.* Automatic Classification for the Identification of Relationships in a Metadata Repository.

9/2003 *Nicholas Kushmerick, Bernd Thomas.* Adaptive information extraction: Core technologies for information agents.

8/2003 *Bernd Thomas.* Bottom-Up Learning of Logic Programs for Information Extraction from Hypertext Documents.

7/2003 *Ulrich Furbach.* AI - A Multiple Book Review.

6/2003 *Peter Baumgartner, Ulrich Furbach, Margret Gro-Hardt.* Living Books.

5/2003 *Oliver Obst.* Using Model-Based Diagnosis to Build Hypotheses about Spatial Environments.

4/2003 *Daniel Lohmann, Jrgen Ebert.* A Generalization of the Hyperspace Approach Using Meta-Models.

3/2003 *Marco Kgler, Oliver Obst.* Simulation League: The Next Generation.

2/2003 *Peter Baumgartner, Margret Gro-Hardt, Alex Sinner.* Living Book – Deduction, Slicing and Interaction.

1/2003 *Peter Baumgartner, Cesare Tinelli.* The Model Evolution Calculus.

2002

12/2002 *Kurt Lautenbach.* Logical Reasoning and Petri Nets.

11/2002 *Margret Gro-Hardt.* Processing of Concept Based Queries for XML Data.

10/2002 *Hanno Binder, Jrme Diebold, Tobias Feldmann, Andreas Kern, David Polock, Dennis Reif, Stephan Schmidt, Frank Schmitt, Dieter Zbel.* Fahrassistenzsystem zur Untersttzung beim Rckwrtsfahren mit einachsigen Gespannen.

9/2002 *Jrgen Ebert, Bernt Kullbach, Franz Lehner.* 4. Workshop Software Reengineering (Bad Honnef, 29./30. April 2002).

8/2002 *Richard C. Holt, Andreas Winter, Jingwei Wu.* Towards a Common Query Language for Reverse Engineering.

7/2002 *Jrgen Ebert, Bernt Kullbach, Volker Riediger, Andreas Winter.* GUPRO – Generic Understanding of Programs, An Overview.

6/2002 *Margret Gro-Hardt.* Concept based querying of semistructured data.

5/2002 *Anna Simon, Marianne Valerius.* User Requirements – Lessons Learned from a Computer Science Course.

4/2002 *Frieder Stolzenburg, Oliver Obst, Jan Murray.* Qualitative Velocity and Ball Interception.

3/2002 *Peter Baumgartner.* A First-Order Logic Davis-Putnam-Logemann-Loveland Procedure.

2/2002 *Peter Baumgartner, Ulrich Furbach.* Automated Deduction Techniques for the Management of Personalized Documents.

1/2002 *Jrgen Ebert, Bernt Kullbach, Franz Lehner.* 3. Workshop Software Reengineering (Bad Honnef, 10./11. Mai 2001).

2001

13/2001 *Annette Pook.* Schlussbericht “FUN - Funkunterrichtsnetzwerk”.

12/2001 *Toshiaki Arai, Frieder Stolzenburg.* Multiagent Systems Specification by UML Statecharts Aiming at Intelligent Manufacturing.

11/2001 *Kurt Lautenbach.* Reproducibility of the Empty Marking.

10/2001 *Jan Murray.* Specifying Agents with UML in Robotic Soccer.

9/2001 *Andreas Winter.* Exchanging Graphs with GXL.

8/2001 *Marianne Valerius, Anna Simon.* Slicing Book Technology — eine neue Technik fr eine neue Lehre?.

7/2001 *Bernt Kullbach, Volker Riediger.* Folding: An Approach to Enable Program Understanding of Preprocessed Languages.

6/2001 *Frieder Stolzenburg.* From the Specification of Multiagent Systems by Statecharts to their Formal Analysis by Model Checking.

5/2001 *Oliver Obst.* Specifying Rational Agents with Statecharts and Utility Functions.

4/2001 *Torsten Gipp, Jrgen Ebert.* Conceptual Modelling and Web Site Generation using Graph Technology.

3/2001 *Carlos I. Chesevar, Jrgen Dix, Frieder Stolzenburg, Guillermo R. Simari.* Relating Defeasible and Normal Logic Programming through Transformation Properties.

2/2001 *Carola Lange, Harry M. Sneed, Andreas Winter.* Applying GUPRO to GEOS – A Case Study.

1/2001 *Pascal von Hutten, Stephan Philippi.* Modelling a concurrent ray-tracing algorithm using object-oriented Petri-Nets.

2000

8/2000 *Jrgen Ebert, Bernt Kullbach, Franz Lehner (Hrsg.).* 2. Workshop Software Reengineering (Bad Honnef, 11./12. Mai 2000).

7/2000 *Stephan Philippi.* AWPN 2000 - 7. Workshop Algorithmen und Werkzeuge fr Petrinetze, Koblenz, 02.-03. Oktober 2000 .

6/2000 *Jan Murray, Oliver Obst, Frieder Stolzenburg.* Towards a Logical Approach for Soccer Agents Engineering.

5/2000 *Peter Baumgartner, Hantao Zhang (Eds.).* FTP 2000 – Third International Workshop on First-Order Theorem Proving, St Andrews, Scotland, July 2000.

4/2000 *Frieder Stolzenburg, Alejandro J. Garca, Carlos I. Chesevar, Guillermo R. Simari.* Introducing Generalized Specificity in Logic Programming.

3/2000 *Ingar Uhe, Manfred Rosendahl.* Specification of Symbols and Implementation of Their Constraints in JKogge.

2/2000 *Peter Baumgartner, Fabio Massacci.* The Taming of the (X)OR.

1/2000 *Richard C. Holt, Andreas Winter, Andy Schrr.* GXL: Towards a Standard Exchange Format.

1999

10/99 *Jrgen Ebert, Luuk Groenewegen, Roger Sttenbach.* A Formalization of SOCCA.

9/99 *Hassan Diab, Ulrich Furbach, Hassan Tabbara.* On the Use of Fuzzy Techniques in Cache Memory Managament.

8/99 *Jens Woch, Friedbert Widmann.* Implementation of a Schema-TAG-Parser.

7/99 *Jrgen Ebert, and Bernt Kullbach, Franz Lehner (Hrsg.).* Workshop Software-Reengineering (Bad Honnef, 27./28. Mai 1999).

6/99 *Peter Baumgartner, Michael Khn.* Abductive Coreference by Model Construction.

5/99 *Jrgen Ebert, Bernt Kullbach, Andreas Winter.* GraX – An Interchange Format for Reengineering Tools.

4/99 *Frieder Stolzenburg, Oliver Obst, Jan Murray, Bjrn Bremer.* Spatial Agents Implemented in a Logical Expressible Language.

3/99 *Kurt Lautenbach, Carlo Simon.* Erweiterte Zeitstempelnetze zur Modellierung hybrider Systeme.

2/99 *Frieder Stolzenburg.* Loop-Detection in Hyper-Tableaux by Powerful Model Generation.

1/99 *Peter Baumgartner, J.D. Horton, Bruce Spencer.* Merge Path Improvements for Minimal Model Hyper Tableaux.