Exchanging Graphs with GXL*

Andreas Winter

Universität Koblenz-Landau, Institut für Softwaretechnik
D-56016 Koblenz, Postfach 201602
mailto:winter@uni-koblenz.de
http://www.gupro.de/winter

Abstract. GXL (Graph eXchange Language) is designed to be a standard data exchange format for graph-based tools. GXL is defined as an XML sublanguage, which offers support for exchanging instance graphs together with their appropriate schema information in a uniform format. Formally, GXL is based on typed, attributed, directed, ordered graphs which are extended by concepts to represent hypergraphs and hierarchical graphs. Using this general graph model, GXL offers a versatile support for exchanging nearly all kinds of graphs.

1 Motivation and Background

A great variety of software tools relies on graphs as internal data representation. A standardized language for exchanging those graphs offers a first step in improving *interoperability* between these tools. In software reengineering, for instance, various graph-based tools are used. These include *extractors* (e. g. scanner, parser), *abstractors* (e. g. query tools, structure recognition tools, slicing tools etc.), and *visualizer* (e. g. graph and diagram visualizer, code browser). Currently, these tool components are used more or less independently. [29] gives an overview on existing combinations of tools used in various reengineering projects. Using a common graph interchange format, these tools can be composed to build a genral and powerful reengineering workbench.

The development of *GXL (Graph eXchange Language)* originally started to support data interoperability between reengineering tools. But since GXL was developed as a general format for describing graph structures, it is applicable in further areas of tool interoperability. Especially, GXL is used to support interoperability between graph transformation systems [45] or graph visualization systems. Now, the work on GXL aims at offering a *general exchange format for graph-based tools.*

Exchanging graphs with GXL deals with both, *instance graphs* and their corresponding *graph schemas.* Firstly, GXL offers a versatile support for exchanging all kinds of data based on *typed, attributed, directed, ordered graphs* including *hypergraphs* and *hierarchical graphs.* Secondly, GXL offers means for exchanging graph schemas representing the graph structure i. e. the definition of node and edge classes, their attribute schemas and their incidence structure. Both, instance graphs and graph schemas, are exchanged by XML documents (Extended Markup Language) [47].

* © Springer Verlag: P. Mutzel (ed.) Graph Drawing - 9th Interational Symposium, GD 2001, Vienna, September 23-26, 2001, Mathematics and Vizualization.

After a short survey of the genealogy of GXL in section 2, the major concepts of GXL to exchange instance graphs are introduced in section 3. The language definition of GXL is given by its XML document type definition (DTD) in section 3.4. Section 4 describes the exchange of graph schemas. The current and intended usage of GXL is summarized in section 5.

More information on GXL can be found in [29] and [27]. Up-to-date information including tutorials and further GXL documents are collected at `http://www.gupro.de/GXL`.

2 Genealogy of GXL

GXL originated in a merger of *GRAph eXchange format (GraX)* [10], *Tuple Attribute Language (TA)* [26], and the graph format of the *PROGRES* graph rewriting system [41]. The graph model resulting from this merger was supplemented by additional concepts to handle hierarchical graphs and hypergraphs. Furthermore, GXL includes ideas from common exchange formats used in reengineering, including *ATerms* [46], *Relation Partition Algebra (RPA)* [36], and *Rigi Standard Format (RSF)* [52]. Further features from XML-based exchange of graph transformation systems, developed by groups in Barcelona, Berlin, Budapest, and Kent [22] were included into GXL. The development of GXL was also influenced by various formats used in graph drawing e. g. *daVinci* [13], *GML/Graphlet* [18], *GRL* [33] *XGMML* [53], and *GraphXML* [25]. Thus, GXL covers most of the important graph formats. It can be viewed as a generalization of these formats. The genealogy of GXL is depicted in figure 2.

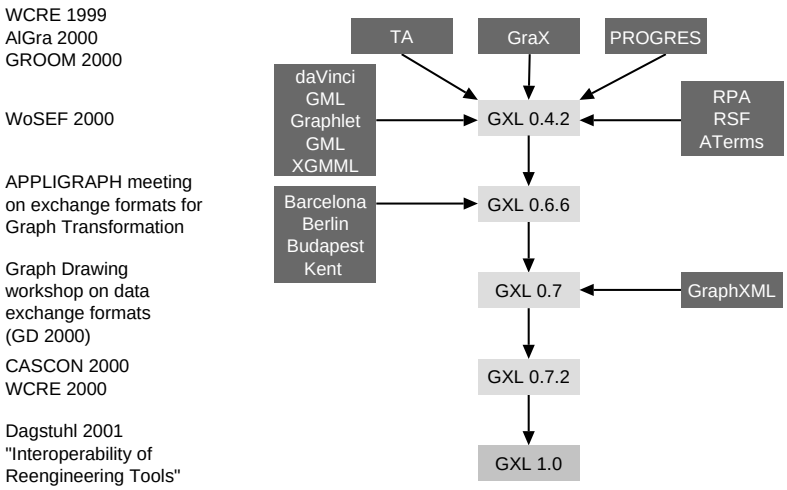

Fig. 1. Genealogy of GXL

The development of GXL was advanced during various conferences and workshops since 1998. First efforts on defining a general exchange format for reengineering data were made at WCRE 1998 [49] and at CASCON 1998 [5]. Ap-

proaches for graph-based exchange formats were discussed during meetings at WCRE 1999 [50], AlGra 2000 [1], and GROOM 2000 [20]. These discussions resulted in the first version of GXL, which was presented at the ICSE 2000 workshop on standard exchange formats (WoSEF 2000) [43]. Subsequent versions were discussed and compared to similar approaches from related areas at the APPLIGRAPH meeting for exchange formats for graph transformation systems [22] and the Graph Drawing 2000 workshop on exchange formats [16]. Improvements of these versions were presented in CASCON 2000 tutorials [28] and workshops [44] and during the WCRE 2000 exchange formats workshop [31].

GXL (version 1.0) was ratified as *standard exchange format* in software reengineering at the Dagstuhl Seminar "Interoperability of Reengineering Tools" in January 2001 [7]. Current work deals with gathering experiences with GXL version 1.0 and providing tool support for working with GXL.

3 Exchanging Graphs

Due to their mathematical foundation and algorithmic power, graphs are a common data structure in software engineering. Different graph models e. g. directed graphs, undirected graphs, node attributed graphs, edge attributed graphs, node typed graphs, edge typed graphs, ordered graphs, relational graphs, acyclic graphs, trees, etc. or combinations of these graph models are utilized in many software systems. To support interoperability of graph-based tools, the underlying graph model has to be as rich as possible to cover most of these graph models.

Such a common graph model is given by *typed, attributed, directed, ordered graphs (TGraphs)* [9], [10]. TGraphs are *directed* graphs, whose nodes and edges may be *attributed* and *typed*. Each type can be assigned an individual attribute schema specifying the possible attributes of nodes and edges. Furthermore, TGraphs are *ordered*, i. e. the node set, the edge set, and the sets of edges incident to a node have a total ordering. This ordering gives modeling power to describe sequences of objects (e. g. parameter lists) and facilitates the implementation of deterministic graph algorithms. In applying TGraphs to the other graph models, not all properties of TGraphs have to be used to their full extent. These graph models can be viewed as specializations of TGraphs. Exchanging typed, attributed, directed, ordered graphs or their specializations with GXL is introduced in section 3.1

To offer support for *hypergraphs* and *hierarchical graphs*, TGraphs were extended by n-ary edges and by nodes and edges containing lower level graphs. GXL language constructs for exchanging hypergraphs and hierarchical graphs are sketched in section 3.2 and 3.3. The complete GXL language definition is given in section 3.4 in terms of a XML document type definition.

3.1 Exchanging typed, attributed, directed, ordered graphs

The object diagram (cf. [40]) in figure 2 shows a node and edge typed, node and edge attributed, directed, ordered graph representing a program fragment on

ASG (abstract syntax graph) level. Function *main* calls function $a = max(a, b)$ in line *8* and function $b = min(b, a)$ in line *19*. The functions *main*, *max* and *min* are represented by nodes of type *Function*. These nodes are attributed with the function name. *FunctionCall* nodes represent the calls of functions *max* and *min*. They are associated to the caller by *isCaller* edges and to the callee by *isCallee* edges. *isCaller* edges are attributed with a line attribute showing the line number which contains the call. Input parameters (represented by *Variable* nodes that are attributed with the variable name) are associated by *isInput* edges. The ordering of parameter lists is given by ordering the incidences of *isInput* edges pointing to *FunctionCall* nodes. The first edge of type *isInput* incident to function call *v2* (modeling the call of *max(a,b)*) comes from node *v6* representing variable *a*. The second edge of type *isInput* connects to the second parameter *b* (node *v7*). The incidences of *isInput* edges associated with node *v3* model the reversed parameter order. Output parameters are associated to their function calls by *isOutput* edges.

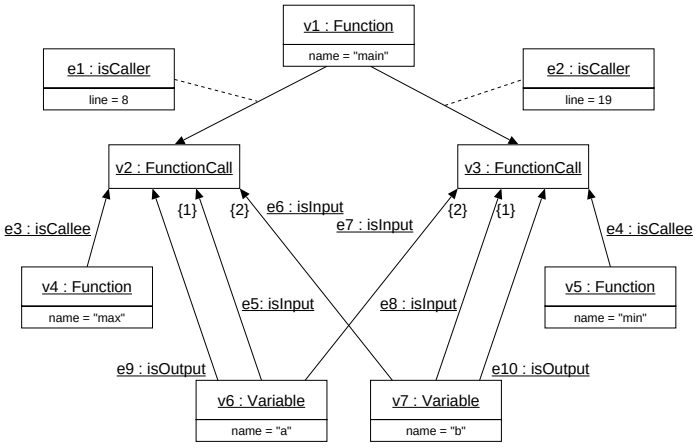

Fig. 2. typed, attributed, directed, ordered graph

Exchanging graphs like the one in figure 2 requires language constructs for representing nodes, edges and their incidence relation. Furthermore, support for describing type information, attribute values, and ordering information is needed.

Figure 3 depicts the graph from figure 2 as GXL document. The complete grammar for these documents is given in section 3.4. XML documents start with specifying the XML version and the underlying document type definition, here "gxl.dtd". The body of a GXL document is enclosed in <**gxl**> tags. The GXL document in figure 3 contains one graph, enclosed in <**graph**> tags, with an unique identifier "simpleGraph". The graph refers to its associated graph schema (cf. section 4) stored in file schema.gxl. GXL supports both, graphs with edges having a unique object identifier, and graphs with unnamed edges. The attribute edgeids = "true" indicates uniquely named edges.

```
<?xml version = "1.0" ?>
<!DOCTYPE gxl SYSTEM "gxl.dtd" >
<gxl>
<graph id = "simpleGraph"
        edgeids = "true">
  <type xlink:href = "schema.gxl"/>
  <node id = "v1" >
    <type xlink:href =
    "schema.gxl#Function"/>
    <attr name = "name" >
      <string>main</string>
    </attr>
  <node id = "v2" >
    <type xlink:href =
    "schema.gxl#FunctionCall"/>
  </node>
  <node id = "v3" >
    <type xlink:href =
    "schema.gxl#FunctionCall"/>
  </node>
  <node id = "v4" >
    <type xlink:href =
    "schema.gxl#Function"/>
    <attr name = "name" >
      <string>max</string>
    </attr>
  </node>
  <node id = "v5" >
    <type xlink:href =
    "schema.gxl#Function"/>
    <attr name = "name" >
      <string>min</string>
    </attr>
  </node>
  <node id = "v6" >
    <type xlink:href =
    "schema.gxl#Variable"/>
    <attr name = "name" >
      <string>a</string>
    </attr>
  </node>
  <node id = "v7" >
    <type xlink:href =
    "schema.gxl#Variable"/>
    <attr name = "name" >
      <string>b</string>
    </attr>
  </node>
  <edge id = "e1"
    from = "v2" to = v1"/>
    <type xlink:href =
    "schema.gxl#isCaller"/>
    <attr name = "line" >
      <int>8</int>
    </attr>
  </edge>
  <edge id = "e2"
    from = "v3" to = v1"/>
    <type xlink:href =
    "schema.gxl#isCaller"/>
    <attr name = "line" >
      <int>19</int>
    </attr>
  </edge>
  <edge id = "e3"
    from = "v4" to = v2"/>
    <type xlink:href =
    "schema.gxl#isCallee"/>
  </edge>
  <edge id = "e9"
    from = "v6" to = v2"
    <type xlink:href =
    "schema.gxl#isOutput"/>
  </edge>
  <edge id = "e5"
    from = "v6" to = v2"
    toorder = "1"/>
    <type xlink:href =
    "schema.gxl#isInput"/>
  </edge>
  <edge id = "e6"
    from = "v7" to = v2"
    toorder = "2"/>
    <type xlink:href =
    "schema.gxl#isInput"/>
  </edge>
  <edge id = "e7"
    from = "v6" to = v3"
    toorder = "2"/>
    <type xlink:href =
    "schema .gxl#isInput"/>
  </edge>
  <edge id = "e8"
    from = "v7" to = v3"
    toorder = "1"/>
    <type xlink:href =
    "schema.gxl#isInput"/>
  </edge>
  <edge id = "e9"
    from = "v6" to = v2"
    <type xlink:href =
    "schema.gxl#isOutput"/>
  </edge>
  <edge id = "e10"
    from = "v7" to = v3"
    <type xlink:href =
    "schema.gxl#isOutput"/>
  </edge>
</graph>
</gxl>
```

Fig. 3. GXL representation of graph from figure 2

Nodes and edges of a given graph are exchanged by <**node**> and <**edge**> elements which can be addressed by their identifier attribute. Incidence information of edges including edge orientation is stored in from and to attributes within <**edge**> tags. Ordering of incidences is also modeled here. Attributes fromorder and toorder represent the position of an edge in the incidence list of its start and target node. Node and edge types are represented by links pointing to the appropriate schema information. This link is enclosed in <**type**> tags.

<**node**> and <**edge**> elements may additionally contain further attribute information. <**attr**> elements describe attribute name and value. Like OCL [48], GXL provides <**bool**>, <**int**>, <**float**>, and <**string**> attributes. Furthermore, enumeration values (<**enum**>) and URI-references (<**locator**>) pointing to externally stored objects are supported. Attribute values might be sub structured. Here, GXL offers composite attributes like sequences (<**seq**>), sets (<**set**>), multi sets (<**bag**>), and tuples (<**tup**>).

3.2 Exchanging Hypergraphs

In addition to graphs, GXL provides the exchange of *hypergraphs*. *Hypergraphs* [3] are graphs with n-ary edges (hyperedges) with arbitrary n. Hyperedges represent

n-ary relations. GXL provides the exchange of typed, attributed, directed and undirected, ordered hypergraphs.

Figure 4 shows a hypergraph in UML notation, modeling the function call $a = max(a, b)$ by a 5-ary hyperedge of type *FunctionCall2*. The diamond, representing the hyperedge, is connected by undirected lines (tentacles) to its related *Function*- and *Variable*-nodes. These tentacles are marked with roles, identifying *caller*, *callee*, *input*, and *output*. Numbers describing the order of incidences of tentacles according the hyperedge, indicate the ordering of parameters. Like edge *e1* in figure 2, the hyperedge is attributed with a *line* attribute.

The GXL representation of this hyperedge is given in figure 5. Hyperedges are represented by <**rel**> elements (relation). Like <**node**> and <**edge**> elements, <**rel**> elements can contain type (<**type**>) and attribute (<**attr**>) information. Tentacles, which point to the related graph objects (target), are represented by <**relend**> subelements (relation end). Roles of tentacles are stored in role attributes. Incidences according to the hyperedge are exchanged by startorder attributes. The ordering of tentacles according their target objects can be modeled by endorder attributes. Directed or undirected hyperedges and tentacles are distinguished by attributes isdirected and direction (cf. the GXL DTD in section 3.4).

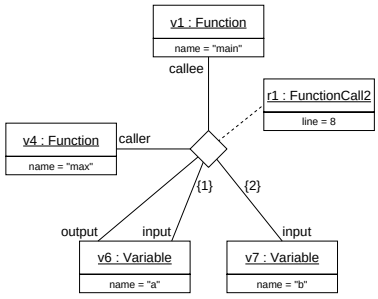

Fig. 4. Hypergraph

```
...
<rel id = "r1" >
   <type xlink:href = "schema2.gxl#FunctionCall2"/>
   <attr name = "line" >
      <int>8</int>
   </attr>
   <relend target = "v1" role = "caller"/>
   <relend target = "v2" role = "callee"/>
   <relend target = "v6" role = "output"/>
   <relend target = "v6" role = "input"
           startorder = "1"/>
   <relend target = "v7" role = "input"
           startorder = "2"/>
</rel>
...
```

Fig. 5. GXL representation

Edges can be viewed as 2-ary hyperedges. Thus, in GXL, edge information can be represented by *binary hyperedges*. Since graphs with (binary) edges are widespread in software engineering and most applications deal with graphs instead of hypergraphs, GXL offers both, the element <**rel**> for hyperedges, and, as a shortcut for binary hyperedges, <**edge**> elements.

3.3 Exchanging Hierarchical Graphs

Graphs gather their popularity from their mathematical foundation and their visual capabilities to express complex contexts. However, due to their size, large graphs become bulky and difficult to understand. This complexity can be reduced by structuring graphs. Parts of graphs representing related objects can be grouped together to form encapsulated, higher level structures. *Hierarchical graphs* [4] support structuring graphs by grouping and encapsulation.

Figure 6 depicts a hierarchical graph. Node *v4* (cf. figure 2) contains a subgraph of type *asg* (abstract syntax graph) representing the implementation

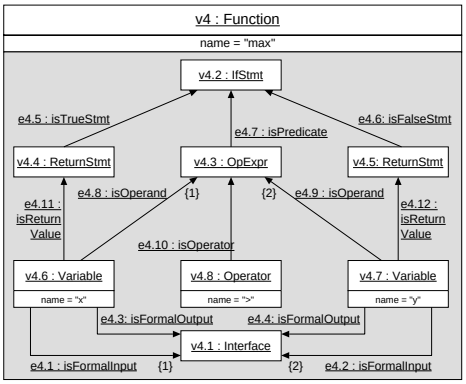

Fig. 6. Hierarchical Graph

```
...
<node id = "v4" >
  <type xlink:href = "schema.gxl#Function" />
  <attr name = "name" >
    <string>max</string>
  </attr>
  <graph id = "max" >
    <type xlink:href = "asg.gxl" >
    <node id = "v4.1" >
      <type xlink:href = "asg.gxl#Interface" >
    </node>
    ...
    <edge id = "e4.12"
      from = "v4.7" to = v4.5"/>
      <type xlink:href =
        "asg.gxl#isReturnValue"/>
    </edge>
  </graph>
</node>
...
```

Fig. 7. GXL representation

of function *max*. The GXL representation in figure 7 shows this subgraph as <**graph**> subelement of node *v4*. Subgraphs associated to edges or hyperedges are exchanged analogously (cf. the GXL DTD in section 3.4).

This representation for hierarchical graphs works for those hierarchical graphs with strong ownership for each graph object. This representation also permits edges and hyperedges crossing the boundaries of graph hierarchies. Those edges are contained in the least-common-ancestor-graph.

Since no general model for hierarchical graphs exists so far (cf. [4]), GXL provides further support for exchanging graph hierarchies. Alternatively, references to subgraphs and their elements can be exchanged using <**locator**> attributes pointing to their appropriate GXL representation. Further support might be offered in a next GXL version by graph-valued attributes or by special edges, representing hierarchy.

3.4 GXL Document Type Definition

The language features of GXL for exchanging typed, attributed, directed, ordered graphs, hypergraphs, and hierarchical graphs are summarized in a conceptual model defining the graph model supported by GXL. The GXL graph model is completely described at `http://www.gupro.de/GXL/` (graph model) with its graph structure part and its attribute part.

Since GXL is a XML sublanguage, the GXL graph model had to be transcribed into a XML document type definition (DTD) or an appropriate XML schema definition. To keep GXL simple and less verbose, this translation was done manually. The resulting DTD (cf. figure 8, a commented version is given at `http://www.gupro.de/GXL` (DTD)) requires only 18 XML elements. In contrast, an appropriate DTD generated with IBMs XMI Toolkit [30] according the XML Metadata Interchange (XMI) principles for developing DTDs [35, section 3] requires 66 elements for the GXL core and and additional 63 elements for XMI and Corba related aspects.

```
<!– extensions –>
<!ENTITY % gxl-extension       "" >
<!ENTITY % graph-extension     "" >
<!ENTITY % node-extension      "" >
<!ENTITY % edge-extension      "" >
<!ENTITY % rel-extension       "" >
<!ENTITY % value-extension     "" >
<!ENTITY % relend-extension    "" >
<!ENTITY % gxl-attr-extension  "" >
<!ENTITY % graph-attr-extension"" >
<!ENTITY % node-attr-extension"" >
<!ENTITY % edge-attr-extension"" >
<!ENTITY % rel-attr-extension  "" >
<!ENTITY % relend-attr-extension"" >

<!– attribute values –>
<!ENTITY % val " locator | bool | int | float | string |
                 enum | seq | set | bag | tup
                 % value-extension;" >

<!– gxl –>
<!ELEMENT gxl (graph* %gxl-extension;) >
<!ATTLIST gxl
    xmlns:xlink CDATA          #FIXED
                               "www.w3.org/1999/xlink"
    %gxl-attr-extension; >

<!– type –>
<!ELEMENT type EMPTY>
<!ATTLIST type
    xlink:type   (simple)      #FIXED "simple"
    xlink:href   CDATA         #REQUIRED >

<!– graph –>
<!ELEMENT graph (type? , attr* ,
                 ( node | edge | rel )*
                 %graph-extension;) >
<!ATTLIST graph
    id           ID            #REQUIRED
    role         NMTOKEN       #IMPLIED
    edgeids      ( true | false )  "false"
    hypergraph   ( true | false )  "false"
    edgemode     ( directed | undirected |
                 defaultdirected | defaultundirected)
                 "directed"
    %graph-attr-extension; >

<!– node –>
<!ELEMENT node (type? , attr*, graph*
                 %node-extension;) >
<!ATTLIST node
    id           ID            #REQUIRED
    %node-attr-extension; >

<!– edge –>
<!ELEMENT edge (type?, attr*, graph*
                 %edge-extension;) >
<!ATTLIST edge
    id           ID            #IMPLIED
    from         IDREF         #REQUIRED
    to           IDREF         #REQUIRED
    fromorder    CDATA         #IMPLIED
    toorder      CDATA         #IMPLIED
    isdirected   ( true | false )  #IMPLIED
    %edge-attr-extension; >

<!– rel –>
<!ELEMENT rel (type? , attr*, graph*, relend*
                 %rel-extension;) >
<!ATTLIST rel
    id           ID            #IMPLIED
    isdirected   ( true | false )  #IMPLIED
    %rel-attr-extension; >

<!– relend –>
<!ELEMENT relend (attr* %relend-extension;) >
<!ATTLIST relend
    target       IDREF         #REQUIRED
    role         NMTOKEN       #IMPLIED
    direction    ( in | out | none)  #IMPLIED
    startorder   CDATA         #IMPLIED
    endorder     CDATA         #IMPLIED
    %relend-attr-extension; >

<!– attr –>
<!ELEMENT attr (type?, attr*, (%val;)) >
<!ATTLIST attr
    id           IDREF         #IMPLIED
    name         NMTOKEN       #REQUIRED
    kind         NMTOKEN       #IMPLIED >

<!– locator –>
<!ELEMENT locator EMPTY >
<!ATTLIST locator
    xlink:type   (simple)      #FIXED "simple"
    xlink:href   CDATA         #IMPLIED >

<!– attribute values –>
<!ELEMENT bool (#PCDATA) >
<!ELEMENT int (#PCDATA) >
<!ELEMENT float (#PCDATA) >
<!ELEMENT string (#PCDATA) >
<!ELEMENT enum (#PCDATA) >
<!ELEMENT seq (%val;)* >
<!ELEMENT set (%val;)* >
<!ELEMENT bag (%val;)* >
<!ELEMENT tup (%val;)* >
```

Fig. 8. GXL Document Type Definition

4 Exchanging Graph Schemas

Graphs only offer a plain structured means for describing objects (nodes) and their interrelationship (edges, hyperedges). Graphs have no meaning of their own. The meaning of graphs corresponds to the context in which they are used and exchanged. The application and interchange context determines

- which node, edge, and hyperedge classes are used,
- which relations exist between nodes, edges, and hyperedges of given classes,
- which attribute structures are associated to nodes, edges, and hyperedges,

– which graph hierarchies are supported, and
– which additional constraints (like ordering of incidences, degree-restrictions etc.) have to be complied.

This schematic data can be described by *conceptual modeling techniques*. Class diagrams offer a suited declarative language to define graph classes with respect to a given application or interchange context [10].

4.1 Describing graph classes by class diagrams

In GXL graph classes are defined by UML class diagrams [40]. Figure 9 shows a graph schema defining classes of graphs like the one given in figure 2. Node classes (*FunctionCall*, *Function*, and *Variable*) are defined by classes. Edge classes (*isCallee*, *isInput*, and *isOutput*) are defined by associations. Attributed edge classes (*isCaller*) are described by associated classes. Like classes, they contain the associated attribute structures. The orientation of edges is depicted by a filled triangle (cf. [40, p. 155]. Since most of the available UML tools do not offer this UML construct, directed arrows (depicting visibility in original UML) can be used alternatively (cf. figure 13). Multiplicities denote degree restrictions. Ordering of incidences is indicated by the keyword {ordered}.

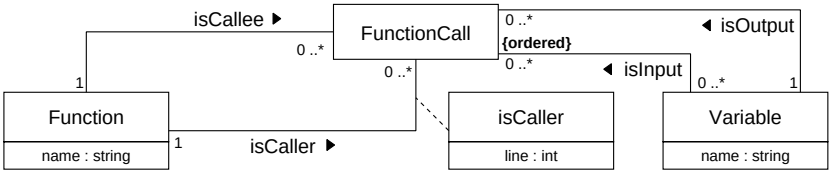

Fig. 9. Graph - Schema

In a similar way, UML class diagrams offer language constructs to specify classes of hypergraphs and hierarchical graphs. Figure 10 shows a class diagram defining hypergraphs like the one in figure 4. Classes of hyperedges are defined by n-ary associations depicted by a diamond. This diamond is connected by links to the related node classes. These links can be annotated by multiplicity information to demand cardinalities, and by names indicating the role of participating classes. The keyword {ordered} demands ordering of tentacles in appropriate instance graphs.

The definition of hierarchical graphs in UML requires an additional language construct representing graph classes themselves. The stereotype <<GraphClass>> distinguishes classes representing graph classes from classes defining node classes. <<GraphClasses>> compose classes and associations to define the graph schema on the next level of hierarchy. UML provides a nested notation to represent these subschemas within the <<GraphClass>> classes. Strong ownership of subgraphs to graph elements is expressed by composition (filled diamond). Figure 11 defines a graph class for hierarchical graphs like the one, depicted in figure 6. Nodes of Class Function contain graphs of graph class *asg*.

To offer up-to-date conceptual modeling power, the GXL schema notation also provides generalization of node-, edge-, and hyperedge classes as well as

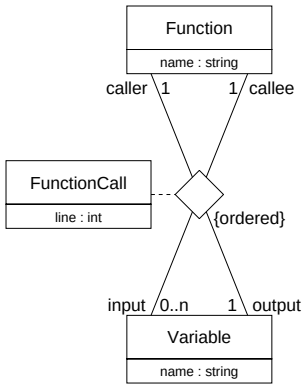

Fig. 10. Hypergraph-Schema

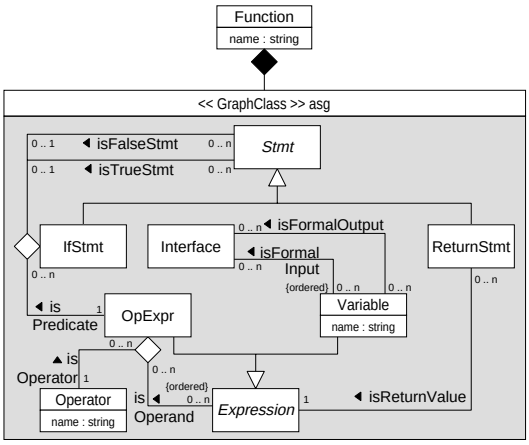

Fig. 11. Hierarchical Graph-Schema

aggregation and composition by using the appropriate UML notation (cf. the definition of <<GraphClass>> *asg* in figure 11.

4.2 Describing graph classes by graphs

Since UML class diagrams are structured information themselves, they may be represented as graphs as well. For exchanging graph schemas in GXL, UML class diagrams are transfered into equivalent graph representations. Thus, instance graphs and schemas are exchanged with the *same type of document*, i. e. XML documents matching the GXL DTD (cf. section 3.4).

Figure 12 depicts the transformation of the class diagram in figure 9 into a node and edge typed, node and edge attributed, directed graph. Node-, edge- and hyperedge-classes, attributes and their domains are modeled by nodes of suitable node types. Their attributes describe further properties. Interrelationships between surrogates of these classes are represented by edges of proper types. Attribute information is associated with surrogates of node, edge, and hyperedge classes and associations by *hasAttribute* and *hasDomain* edges. *from* and *to* edges model incidences of associations including their orientation. Multiplicities of associations are stored in limits-attributes. The boolean attribute isOrdered indicates ordered incidences.

GXL documents, representing instance graphs to a given graph schema, refer to those nodes of the equivalent schema graph representing node classes (*NodeClass*), edge classes (*EdgeClass*) and hyperedge classes (*RelClass*). The nodes representing these class definitions in figure 12 which are referred by the graph in figure 2 are shaded.

Class diagrams defining hypergraphs or hierarchical graphs can be transformed into graphs analogously. Each class diagram defining a GXL graph schema can be transformed into a graph (schema graph) matching a suited schema, representing GXL schema graphs. Schema graphs are instances of the *GXL metaschema* (cf. section 4.3). They are exchanged like all instance graphs (cf.

section 3) referring to a GXL schema graph. Since the schema graph, representing the GXL metaschema, is an instance of itself, it is exchanged by a self referring GXL document.

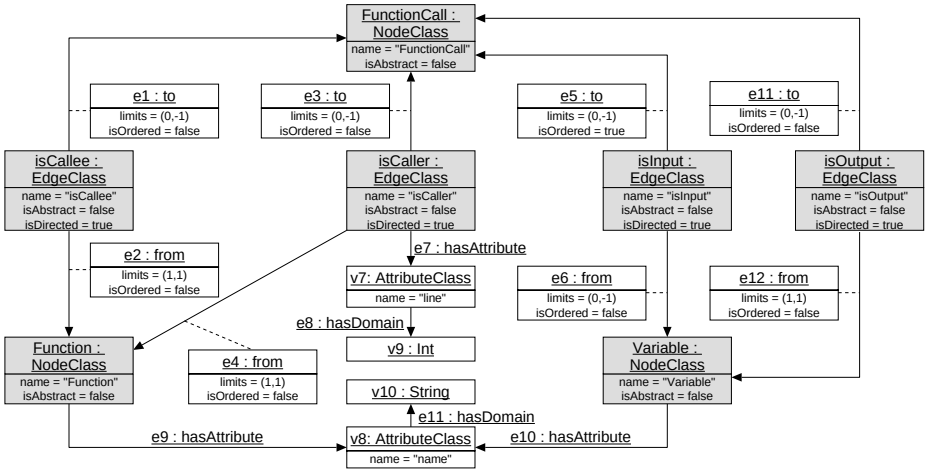

Fig. 12. Graph - Schema (schema graph)

In contrast to the strategy proposed by XML Meta Data Interchange (XMI) [35], GXL schemas are *not* exchanged by XML documents according to the Meta Object Facility (MOF) [34]. XMI/MOF offers a general, but very verbose format for exchanging UML class diagrams as XML streams. Next to its exaggerated verbosity, which contradicts the requirement for exchange formats of as compact as possible documents, the XMI/MOF approach requires *different types of documents* for representing schema and instance graphs. Especially in applications dealing with schema information on instance level (e. g. in tools for editing and analyzing schemas), this leads to the disadvantage of different documents representing the same information, one on instance level (as XML document) and one on schema level (as XML DTD). The GXL approach treats schema and instance information in exactly the same way. Schema and instance graphs are exchanged according the same DTD given in figure 8.

4.3 GXL Metaschema

The GXL metaschema defines the set of graphs representing correct GXL schema graphs. The class diagram in figure 13 shows the graph part of the GXL metaschema. This graph class provides constructs to define classes of graph elements i. e. nodes (NodeClass), edges(EdgeClass), and hyperedges (RelClass) including their interrelationships. Their attributes distinguish abstract classes, classes of directed or undirected edges or hyperedges or contain multiplicity constraints etc.

To express associated attribute structures GraphElementClasses can be connected to attribute structures (AttributedElementClass). The definition of attribute structures supports the structured attributes used in GXL including the definition of default values (cf. the complete GXL metaschema, including its attribute

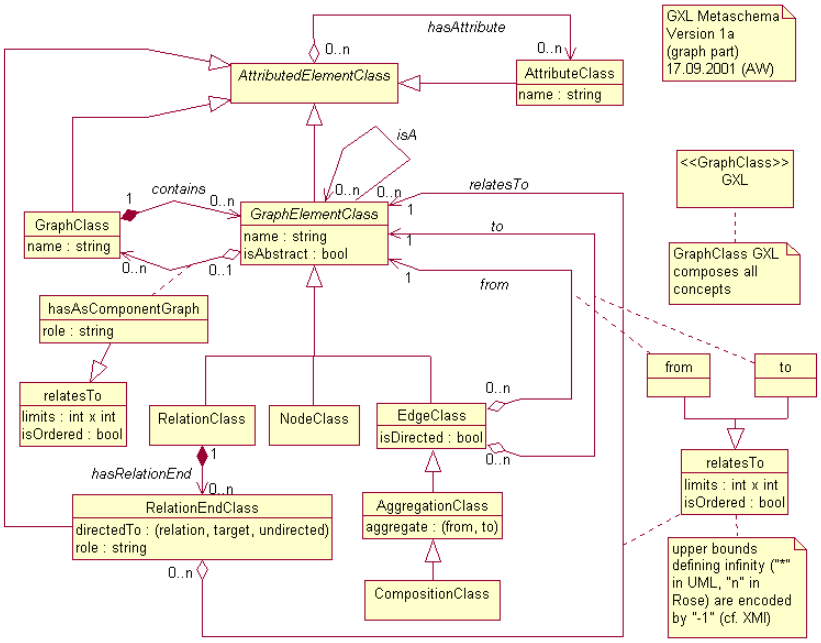

Fig. 13. GXL Metaschema (graph part)

and value parts, at `http://www.gupro.de/GXL/` (meta schema)). Generalization is provided for all GraphElementClasses by isA edges. GraphElementClasses containing lower level graphs are associated to the representation of the lower level GraphClass representation by contains edges. The GraphClass contains those node-, edge-, and hyperedge classes representing its structure. Aggregation (AggregationClass) and composition (CompositonClass) are modeled by specializations of EdgeClasses.

5 Using GXL

At the Dagstuhl seminar on "Interoperability of Reengineering Tools" GXL version 1.0 was ratified as the *standard exchange format* in software reengineering [7]. Currently, various groups in software (re)engineering are implementing GXL import and export facilities to their tools (e. g. Bauhaus [2], Columbus [11], CPPX [6], Fujaba [14], GUPRO [23], PBS [37], RPA (Philips Research), PROGRES [38], Rigi [39], Shrimp [42]). Others are going to implement tools to support working with GXL. For instance, a framework for GXL Converters [15] and a XMI2GXL translator [54] are being developed at Univ. BW München. Further activities deal with providing graph query machines (GReQL, Univ. Koblenz) to GXL graphs or GXL-based graph databases (Univ. Aachen).

An important feature of GXL is its support for exchanging schema information. Based on this capability, reference schemas for certain standard applications in reengineering are currently under development. These activities address reference schemas for data reverse engineering (DRE, Univ. Namur, Paderborn,

Victoria), the Dagstuhl Middle Model [32] or abstract syntax graph models for C++ [6], [12].

Furthermore, groups developing graph transformation tools (e. g. GenSet [17], PROGRES [38]) or graph visualization tools (e. g. GVF [24], Shrimp [42], yFiles [55]) already use GXL or pronounced to use GXL. At University of Toronto, GXL is applied within an undergraduate software engineering course to create a graph editor/layouter [8].

GXL also serves as foundation to define further graph oriented exchange formats. Thus, GXL defines the graph part in the exchange format GTXL (Graph Transformation eXchange Language) [21], [45]. Activities in the graph drawing community also deal with the development of an exchange format for graphs including layout information [16]. There is evidence of combining the structure part of GXL with the graph layout part and the modularization part of GraphML [19] to form a general and comprehensive graph exchange format.

6 Conclusion

The previous sections gave a short introduction in the GXL Graph eXchange Language version 1.0 and its current application.

Summarizing, GXL offers an already widely used XML sublanguage for interchanging typed, attributed, directed, ordered graphs including hypergraphs and hierarchical graphs together with their appropriate schemas. By focusing on graph structure, GXL contributes the core for defining a family of special suited graph exchange formats.

Acknowledgment. I would like to thank the GXL co-authors Richard C. Holt, Andy Schürr, and Susan Elliott Sim for various fruitful discussions on the development of GXL, Jürgen Ebert, Bernt Kullbach, and Volker Riediger for many interesting discussions on TGraphs and GXL, and Kevin Hirschmann for implementing the GUPRO related GXL tools. Thanks to all users of GXL, who currently applying and testing GXL 1.0 in their tools. Their experience will be an important aid to improve GXL.

References

1. Workshop on Algebraic and Graph-Theoretic Approaches in Software Reengineering, Koblenz, February 28, 2000. `http://www.uni-koblenz.de/~winter/AlGra/algra.html` (14.9.2001).
2. Bauhaus: Software Architecture, Software Reengineering, Program Understanding. `http://www.informatik.uni-stuttgart.de/ifi/ps/bauhaus/` (1.9.2001).
3. C. Berge. *Graphs and Hypergraphs.* North-Holland, Amsterdam, 2 edition, 1976.
4. G. Busatto. An Abstract Model of Hierarchical Graphs and Hierarchical Graph Transformation (current draft). `http://www.informatik.uni-bremen.de/~giorgio/papers/phd-thesis.ps.gz` (16.9.2001).
5. Data Exchange Group, Conclusions from Meeting at CASCON 1998,30. Nov 1998. `http://plg.uwaterloo.ca/~holt/sw.eng/exch.format` (14.9.2001).

6. CPPX: Open Source C++ Fact Extractor. http://swag.uwaterloo.ca/~cppx/ (1.9.2001).
7. J. Ebert, K. Kontogiannis, J. Mylopoulos: Interoperability of Reengineering Tools. http://www.dagstuhl.de/DATA/Reports/01041/ (18.4.2001), 2001.
8. S. Easterbrook. CSC444F: Software Engineering I (Fall term 2001), University of Toronto. http://www.cs.toronto.edu/~sme/CSC444F/ (15.9.2001), 2001.
9. J. Ebert and A. Franzke. A Declarative Approach to Graph Based Modeling. In E. Mayr, G. Schmidt, and G. Tinhofer, editors. *Graphtheoretic Concepts in Computer Science, LNCS 903*. Springer, Berlin, pages 38–50. 1995.
10. J. Ebert, B. Kullbach, and A. Winter. GraX – An Interchange Format for Reengineering Tools. In *[50]*, pages 89–98. 1999.
11. R. Ferenc, F. Magyar, Á. Beszédes, Á. Kiss, and M. Tarkiainen. Columbus - Tool for Reverse Engineering Large Object Oriented Software Systems. In *Proceedings SPLST 2001, Szeged, Hungary (http://www.inf.u-szeged.hu/~ferenc/research/ferencr_columbus.pdf, (1.9.2001))*, pages 16–27. June 2001.
12. R. Ferenc, S. Elliott Sim, R. C. Holt, R. Koschke, and T. Gyimòthy. Towards a Standard Schema for C/C++. To appear in *8th Working Conference on Reverse Engineering*. IEEE Computer Soc., 2001.
13. M. Fröhlich and M. Werner. daVinci V2.0.x Online Documentation. http://www.tzi.de/~davinci/docs/ (18.4.2001), June 1996.
14. Fujaba: From UML to Java and back again. http://www.uni-paderborn.de/cs/fujaba/ (1.9.2001).
15. GCF - a GXL Converter Framework. http://www2.informatik.unibw-muenchen.de/GXL/triebsees/ (1.9.2001).
16. Workshop on Data Exchange Formats, Graph Drawing 2000. http://www.cs.virginia.edu/~gd2000/gd-satellite.html (14.9.2001), 2001.
17. GenSet: Design Information Fusion. http://www.cs.uoregon.edu/research/perpetual/dasada/Software/GenSet/ (1.9.2001).
18. The GML File Format. http://www.infosun.fmi.uni-passau.de/Graphlet/GML/ (18.4.2001).
19. The GraphML File Format. http://www.graphdrawing.org/graphml/ (31.8.2001), 2001.
20. 7-ter Workshop des GI-Arbeitskreises GROOM, UML - Erweiterungen und Konzepte der Metamodellierung, 4.-5. April 2000, Universität Koblenz-Landau. http://www2.informatik.unibw-muenchen.de/GROOM/META/ (14.9.2001).
21. Graph Transformation System Exchange Language. http://tfs.cs.tu-berlin.de/projekte/gxl-gtxl.html (18.08.2001).
22. First APPLIGRAPH meeting on GXL (graph exchange language) and GTXL (graph transformation exchange language) Paderborn (September 5-6, 2000). http://tfs.cs.tu-berlin.de/projekte/gxl-gtxl/paderborn.html (11.9.2001).
23. GUPRO: Generic Understanding of Programs. http://www.gupro.de/ (1.9.2001).
24. GVF - Graph Visualization Framework. http://www.cwi.nl/InfoVisu (1.9.2001).
25. I. Herman and M. S. Marshall. Graph XML – An XML based graph interchange format. Report INS-0009, CWI, Amsterdam, April 2000.
26. R. C. Holt. An Introduction to TA: The Tuple-Attribute Language. http://plg.uwaterloo.ca/~holt/papers/ta.html (18.4.2001), 1997.
27. R. C. Holt and A. Winter. A Short Introduction to the GXL Software Exchange Format. In *[51]*, pages 299–301. 2000.
28. R. C. Holt and A. Winter. Software Data Interchange with GXL: Introduction and Tutorial, CASCON 2000. http://www.cas.ibm.com/archives/2000/workshops/descriptions.shtml\#16 (15.9.2001), November 13-16, 2000.

29. R. C. Holt, A. Winter, and A. Schürr. GXL: Toward a Standard Exchange Format. In *[51]*, pages 162–171. 2000.
30. XMI Toolkit 1.15. `http://alphaworks.ibm.com/tech/xmitoolkit` (1.9.2001).
31. K. Kontogiannis. Exchange Formats Workshop. In *[51]*, pages 277–301. 2000.
32. T. Lethbridge, E. Plödereder, S. Tichelar, C. Riva, and P. Linos. The Dagstuhl Middle Level Model (DMM). internal note, 2001.
33. F. Newbery Paulish. *The Design of an Extendible Graph Editor*, *LNCS 704*. Springer, Berlin, 1991.
34. Meta Object Facility (MOF) Specification. `http://www.omg.org/technology/documents/formal/mof.htm` (2.9.2001), March 2000.
35. XML Meta Data Interchange (XMI) Specification. `http://www.omg.org/technology/documents/formal/xmi.htm` (1.9.2001), November 2000.
36. R. Ommering, L. van Feijs, and R. Krikhaar. A relational approach to support software architecture analysis. *Software Practice and Experience*, 28(4), pages 371–400, April 1998.
37. PBS: The Portable Bookshelf. `http://swag.uwaterloo.ca/pbs/` (1.9.2001).
38. A Graph Grammar Programming Environment - PROGRES. `http://www-i3.informatik.rwth-aachen.de/research/projects/progres/` (1.9.2001).
39. RIGI: a visual tool for understanding legacy systems. `http://www.rigi.csc.uvic.ca/` (1.9.2001).
40. J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual*. Addison Wesley, Reading, 1999.
41. A. Schürr, A. J. Winter, and A. Zündorf. PROGRES: Language and Environment. In H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg, editors. *Handbook on Graph Grammars*, volume 2. World Scientific, Singapore, pages 487–550. 1999.
42. ShriMP Views: simple Hierarchical Multi-Perspective. `http://www.shrimpviews.com/` (1.9.2001).
43. S. Elliot Sim, R. C. Holt, and R. Koschke. Proceedings ICSE 2000 Workshop on Standard Exchange Format (WoSEF). Technical report, Limerick, 2000.
44. S. Elliott Sim. Software Data Interchange with GXL: Implementation Issues, CASCON 2000. `http://www.cas.ibm.com/archives/2000/workshops/descriptions.shtml\#17` (14.9.2001), November 13-16, 2000.
45. G. Taenzer. Towards Common Exchange Formats for Graphs and Graph Transformation Systems. In *Proceedings UNIGRA satellite workshop of ETAPS'01*. 2001.
46. M. van den Brand, H. A. de Jong, P. Klint, and P. A. Olivier. Efficient annotated Terms. *Software: Practice and Experience*, 30(3), pages 259–291, March 2000.
47. Extensible Markup Language (XML) 1.0. W3c recommendation, W3C XML Working Group, `http://www.w3.org/XML/` (17.4.2001), February 1998.
48. J. B. Warmer and A. G. Kleppe. *The Object Constraint Language : Precise Modeling With UML*. Addison-Wesley, 1998.
49. *5th Working Conference on Reverse Engineering*. IEEE Computer Soc., 1998.
50. *6th Working Conference on Reverse Engineering*. IEEE Computer Soc., 1999.
51. *7th Working Conference on Reverse Engineering*. IEEE Computer Soc., 2000.
52. K. Wong. RIGI User's Manual, Version 5.4.4. `http://www.rigi.csc.uvic.ca/rigi/rigiframe1.shtml?Download` (18.4.2001), 30. June 1998.
53. Extensible Graph Markup and Modeling Language). `http://www.cs.rpi.edu/~puninj/XGMML/` (19.8.2001), 2001.
54. XIG - An XSLT-based XMI2GXL-Translator. `http://ist.unibw-muenchen.de/GXL/volk/` (1.9.2001).
55. yFiles - Interactive Visualization of Graph Strucutres. `http://www-pr.informatik.uni-tuebingen.de/yfiles/` (1.9.2001).