A New Version of GTXL: An Exchange Format for Graph Transformation Systems

Leen Lambers

Institut für Softwaretechnik und Theoretische Informatik
Technische Universität Berlin
Germany

Abstract

GTXL (Graph Transformation Exchange Language) is designed to support and stimulate developers to provide their graph transformation-based tools with an exchange functionality regarding the integration with other tools. For this exchange XML was chosen as underlying technology. The exchange of graphs is facilitated by the exchange format GXL which is also XML-based. GTXL uses GXL to describe the graph parts of a graph transformation system. A first version of GTXL arose from the format discussion within the EU Working Group APPLIGRAPH. Trying to restimulate the discussion on a common exchange format for graph transformation systems, this paper presents a new version of GTXL. Three important changes have been made. At first, an integrated presentation of rules is introduced, secondly the expression of more general conditions is supported and finally the storage of the underlying semantics of a graph transformation system by means of special attributes is proposed.

Keywords: exchange format, graph transformation system, graph transformation-based tools, XML

1 Introduction

Graph transformation systems (GTSs) comprise graphs and rules changing these graphs. Comparing them with term rewrite systems, terms correspond to graphs and term rewrite rules to graph rewrite rules. Since graphs are widely used to model various data structures in computer science, graph transformation systems can be used to define various modifications of these data

1 Email: leen@cs.tu-berlin.de
structures. Application areas of GTSs [4] [5] are e.g. definition of visual languages, software visualization and animation, reengineering, concurrent and distributed systems etc.

The development of a common exchange format for GTSs has been discussed already within the EU Working Group APPLIGRAPH (Applications of Graph Transformation) [19]. A summary of this format discussion has been presented at UNIGRA, a satellite workshop of ETAPS’01 in Genova at the end of March, 2001 [13]. A common exchange format for GTSs though has not been established yet. This paper presents a new version of GTXL trying to restimulate this format discussion.

The structure of the exchange format proposed in this paper should be easily comprehensible for users familiar with graph transformation theory and applications. XML [7] has been chosen as underlying technology of GTXL, because of its broad acceptance and great adaptibility defining exchange formats. Moreover it is convenient to integrate the widely accepted exchange format for graphs GXL [19]. Thus, the work being done in the graph community is being reused and integrated in GTXL. Considering the main remarks on the former format discussion, three important changes have been made in GTXL. The first version of GTXL defines a graph transformation rule by two so-called rule graphs and a mapping between them. The new version supports, as do a lot of graph transformation tools, an integrated presentation of graph transformation rules. Therefore it should be quite comfortable for most of the developers to provide their tool with an export/import functionality to/of GTXL. Secondly, the first version supported only simple conditions on the left or right graph of a graph transformation rule. Now the description of logical expressions over atomic conditions with the logical operations $\land$, $\lor$ and $\neg$ is possible. The difference between positive and negative application conditions is clearer and the expression of constraints on a GTS is facilitated. Finally, this new version of GTXL proposes to use so-called semantic attributes for saving the underlying conceptual approach used by the tool. This is a simple way to determine the operational semantics of the GTS described in a GTXL file. In fact, it will be necessary to discuss and agree on a clear classification of different GTS-approaches s.t. the use of semantic attributes is efficient.

This paper is organized as follows. It starts with a short summarizing section on GXL, since this is an important module of GTXL. An explanation of the overall structure of GTXL follows, emphasizing the differences with the first version of GTXL. Of course, the rules of a GTS play a very important role. Therefore in the next section, the focus is on the structure of the rule part of GTXL. Different kinds of rules like rules with conditions, gluing rules and rules on attributed graphs are explained and illustrated by sample rules.
and GTXL code snippets. The paper concludes with a short summary, open questions and an outlook regarding this format.

2 GXL revisited

Graphs are used to model various data structures in computer science. GXL is a standard exchange format for graphs [8][17]. In particular, GXL was developed to facilitate the interoperability between software reengineering tools and components, such as code extractors (parsers), analyzers and visualizers. That way, single-purpose graph-based tools can be integrated into a powerful reengineering workbench. To realize these requirements, GXL does not only support the exchange of simple graph structures but also of typed, attributed, directed and ordered graphs. Moreover, hypergraphs and hierarchical graphs can be stored in GXL.

A short description of the GXL structure follows. A GXL document consists of a set of graphs where each graph consists of a set of graph elements. Those graph elements can be nodes, edges or relations to describe hyperedges. They are stored respectively in the GXL elements node, edge and rel. Graphs and graph elements are typed elements. The typing information can be retained in a so-called graph schema or type graph described again by GXL. The element type contains a link to this graph schema or type graph which may be found in another document. Thus, GXL can be used to exchange graphs and their corresponding graph schemas representing the graph structure of these graphs. Finally, graph elements may contain attributes. They are stored in the element attr. Please note that attr contains a value part described by the entity val in GXL. This entity combines the attribute types available in GXL.

Since graphs or parts of graphs are components of GTSs, GXL is used as a module for GTXL. The GXL elements graph, type, node, edge, rel, attr and val are directly used in GTXL documents. Both exchange formats are XML-based, so it is technically convenient to include GXL in GTXL. Thus, using GXL as a module, GTXL benefits from the generality of the exchange format GXL and the work already done in the graph community.

3 Structure of GTXL

The graph transformation exchange format GTXL should fulfill the following overall requirements. It should be compact, meaning that redundancy is avoided. Additionally it should be transparent, meaning that the structure of a GTS is reflected as direct as possible. On the other hand, it should
be general, meaning that GTXL is able to describe different kinds of graph transformation techniques. Finally it should be complete, meaning that all the relevant information available on the GTS can be stored. It is not an easy task to incorporate all these properties in one format, therefore they should be carefully balanced. GTXL is designed to fulfill these requirements and accordingly to encourage the automation of exchanging GTSs.

The syntax of a GTXL document is expressed in the DTD (Document Type Definition). For readers familiar with DTD’s it should be easy to recognize the graph transformation structure in the following gtxl.dtd.

```
<!ENTITY % gxl PUBLIC "http://www.gupro.de/GXL/gxl-1.0.dtd" "gxl-1.0.dtd">
%gxl;
<!ELEMENT gtxl (graph|gts)+>
<!ATTLIST gtxl
 xmlns:xlink CDATA #FIXED "http://www.w3.org/1999/xlink">
<!ELEMENT gts (attr*, type?, initial*,
 (rule | transformationUnit)*, condition?)>
<!ATTLIST gts
 approach NMTOKEN #REQUIRED
 name NMTOKEN #IMPLIED>
<!ELEMENT initial (graph)>
<!ELEMENT transformationUnit (EMPTY) <!-- to be defined -->
<!ELEMENT rule (preserved, deleted, created, parameter*,
 variable*, precondition?, postcondition?, embedding?, attr*)>
<!ATTLIST rule
 id ID #IMPLIED
 name NMTOKEN #IMPLIED>
<!ELEMENT preserved (graph, map*)>
<!ELEMENT deleted (node | edge | rel | attr)*>
<!ELEMENT created (node | edge | rel | attr)*>
<!-- to be used for gluing of graph elements -->
<!ELEMENT map (node | edge | rel)>
<!ATTLIST map
 source IDREFS #REQUIRED>
<!ELEMENT precondition (condition)>
<!ELEMENT postcondition (condition)>
<!ELEMENT condition (attrCondition | graphCondition | condition)+>
<!ATTLIST condition
 id ID #IMPLIED
 isConjunction (true | false) "true"
 isPositive (true | false) "true"
 subconditions IDREFS #IMPLIED>
<!ELEMENT graphCondition ((node|edge|rel)*,map*,graphCondition*)>
<!ATTLIST graphCondition
 id ID #IMPLIED
 name NMTOKEN #IMPLIED>
<!ELEMENT attrCondition (#PCDATA)>
<!ATTLIST attrCondition
 id ID #IMPLIED
 name NMTOKEN #IMPLIED>
<!ELEMENT embedding (EMPTY) <!-- to be defined -->
<!ELEMENT parameter (%val;)
<!ATTLIST parameter
 name NMTOKEN #REQUIRED
 type (in|out|inout) #REQUIRED>
<!ELEMENT variable (%val;)
<!ATTLIST variable
 name NMTOKEN #REQUIRED>
```

Of course, the meaning of the elements of such a DTD needs to be specified,
although most of the element names already give a hint to their meaning. Please notice that this paper only tries to explain the main concepts and presents some examples, but is not a complete specification of this DTD. UML class diagrams [15] offer a nice visualization of the syntactical structure of a GTXL document. Therefore, the overall structure of GTXL is presented in the class diagram in Fig. 1 up to subsubclasses of the GTXL class. The rule and condition part of GTXL will be presented in detail in section 4.1.

Analyzing the class diagram in Fig. 1, one can see that a GTXL document consists of a sequence of graphs and GTSs. As mentioned in the introduction, graphs are formatted like in GXL. They can for example contain intermediate graph results or type graphs for the GTSs. Normally only one GTS will be stored in a document. Sometimes though it could be useful to store more than one. The information about the GTS itself is stored in the element gts. This element can have a name and possesses a so-called semantic attribute to store the underlying conceptual approach of the GTS e.g. approach="dpo", approach="spo" or approach="node-replacement". It will be important to discuss within the graph transformation community the exact meaning of the approaches inserted here and how to name them, e.g. "dpo" rather than "double-pushout-approach". Subsequently these options for the value of the approach attribute could be included explicitly in the gtxl.dtd to avoid any confusion. A gts can have several initial graphs, a couple of rules or transformationUnits, a condition on the whole GTS, a reference to a type graph describing the graph structure of the graphs in the GTS (type is a
GXL element) and an arbitrary amount of elements with the name attr (GXL element) to save various tool dependent information e.g. to store which attribution language is used. Please note again that a discussion and agreement on the variety of semantic attributes used here and there exact meaning is necessary. It should be remarked that the element transformationUnit is not yet defined. It should represent a procedural abstraction of graph rules to express a kind of control flow on the rule system [9]. Since a transformationUnit will most likely contain references to rules, it is important that a rule can have an id. Please note that the possibility of expressing a condition on the whole GTS is a new feature of GTXL. Of course the most important one of all the subelements of gts is the element rule. Therefore the whole next section is devoted to the rule part of the GTXL structure, emphasizing the integrated presentation of rules as well as the expresiveness of conditions on rules.

4 The Rule Part of GTXL

4.1 Structure of a Rule

The biggest part of the gtxl.dtd consists of the elaboration of the rule part of a GTS. Like the structure of GTXL, the structure of a rule can be represented by a class diagram as in Fig. 2. Supporting the integrated presentation of rules, a rule consists of a deleted, preserved and created part. Unlike the presentation proposed in the first GTXL version [13] where a rule consists of

Fig. 2. Class diagram of rule part
a left-hand side graph (LHS), a right-hand side graph (RHS) and a mapping from the LHS to the RHS. The choice of the integrated presentation arose from the format discussion held in the 1st GraBaTS-Workshop (2002) [8]. The main reason is the support of the integrated presentation in most of the graph transformation tools. Thus, a translation of their data structure in the GTXL format should be simplified.

The **deleted** part of a rule consists of all the graph elements occurring in the LHS but not in the RHS of a rule. The **created** part on the contrary consists of all those graph elements occurring in the RHS but not in the LHS. The **preserved** part deserves more discussion. As most of the graph transformation tools forbid gluing rules though, this part is mostly easy again. In this case, the **preserved** part of a rule consists of a graph containing all the elements occurring in both the LHS and RHS of the rule. But what happens when the rule is a gluing one? That means, what if the rule maps two or even more elements to the same one? The **map** element of the **preserved** part expresses exactly such a gluing. An example follows in section 4.3.

Two other important child elements of **rule** are the **precondition** and **postcondition** element. They both consist of a **condition** element and express which conditions should be satisfied before or after the application of a graph transformation rule. More precisely a left and right application condition if **condition** is a descendant of the **precondition** resp. **postcondition** element. The structure of a **condition** is represented by a class diagram as in

![Fig. 3. Class diagram of condition](image)

Fig. 3. It is a logical expression over atomic conditions using the logical operations $\land$, $\lor$ and $\neg$. The boolean attributes **isConjunction** and **isPositive** express whether a condition is a conjunction, disjunction or negation of atomic
conditions. By means of the attribute **subconditions** it is possible to refer to a list of id’s of conditions already introduced before, instead of listing explicitly these subconditions once again. Atomic conditions can be **attrCondition** as well as **graphCondition**. The first one expresses a condition on attributes on the LHS or RHS of a rule. The second one expresses graphical left or right application conditions on a rule. Moreover it is possible to express conditional application conditions because of the nesting of the **graphCondition** element. Comparable to gluing rules, it is possible to express gluing conditions with the **map** element.

Additionally, a rule can possess **parameters** and **variables** consisting of the entity **val** as explained in section 2. Where to use parameters and variables is explained in section 4.4. A rule is also allowed to possess attributes, expressing some rule specific additional information. For example the layer [2] of a rule could be expressed by an attribute. Again the exact meaning of the semantic attributes used here should be discussed and agreed on. Finally, a rule can specify an **embedding** [14] into a variable context. This element is still abstract and has to be further specified.

### 4.2 Example

The running example is taken from a simple business model of a shipping company using graph transformation. We show two rules. The first one, **LoadTruck**, is presented in Fig 4. This rule describes the loading of a container

![Fig. 4. LoadTruck](image)

with a truck parked in front of a store. The **in**-edge should be deleted and a new **on**-edge should be created. All the other graph elements are preserved. Moreover there is a negative application condition on the rule, expressing that naturally the rule cannot be applied if there is already a container on the truck. Thus, this rule comprises a lot of the functionalities a rule can have. Fig. 5 shows an integrated presentation of the rule **LoadTruck**. Note that the same
id’s have been used as in Fig. 4. The following code fragment then shows how the rule LoadTruck is translated into GTXL.

```
<rule name="LoadTruck">
  <preserved>
    <node id="n1"><type xlink:href="#Container"/></node>
    <node id="n3"><type xlink:href="#Truck"/></node>
    <node id="n4"><type xlink:href="#Store"/></node>
    <edge from="n3" to="n4">
      <type xlink:href="#inFrontOf"/>
    </edge>
  </preserved>
  <deleted>
    <edge from="n1" to="n4"><type xlink:href="#in"/></edge>
  </deleted>
  <created>
    <edge from="n1" to="n3"><type xlink:href="#on"/></edge>
  </created>
  <precondition>
    <condition isPositive="false">
      <graphCondition name="emptyTruck">
        <node id="n2"><type xlink:href="#Container"/></node>
        <edge from="n2" to="n3"><type xlink:href="#on"/></edge>
      </graphCondition>
      <graphCondition/>
    </condition>
  </precondition>
</rule>
```

Please note that the graphCondition element contains all the elements that should not exist before application of the rule.

### 4.3 Example of a Gluing Rule

As mentioned in Section 4.1, rules which glue graph elements, are a special case. Tools using the integrated presentation usually do not allow this kind of rules. The map element is designed to deal with gluing rules. An example of such a rule is given in Fig. 6. It describes the fusion of two different companies. The following code shows how rule Fusion is translated into GTXL.

```
<rule name="Fusion">
  <preserved>
    <graph id="g1">
      <node id="n1"><type xlink:href="#Comp"/></node>
      <node id="n2"><type xlink:href="#Comp"/></node>
    </graph>
    <map source="n1 n2">
      <node id="n3"><type xlink:href="#Comp"/></node>
    </map>
  </preserved>
</rule>
```
Please note that element map with the attribute source storing a list of id’s of elements of the LHS, maps this list onto the node with id="n3" of the RHS.

4.4 Rules on attributed graphs

The example rules LoadTruck and Fusion presented before are still quite simple. Attributes can be used to store more detailed information like for example the name or location of a company. Various graph transformation based tools operate with rules on attributed graphs. AGG [1] for example, a graph transformation tool developed at the Technical University of Berlin, uses Java as attribution language [6]. It allows the user to define parameters and needs variables as attribute values. So the question is: Can GTXL deal with rules on attributed graphs as well?

Dealing with attributes is not that simple. Imagine a rule preserves an element but changes the value of one of its attributes. It would not be consistent to list the attribute belonging to the preserved element only once with its new value. The GTXL format should express that the attribute value really changed and somewhere the old value of the attribute should be stored. A proposal how to handle this problem is described here shortly.

Attributes of preserved elements changing their value should be listed once with their old value in the deleted part and once with their new value in the created part of the rule. Notice that in the gtxl.dtd the element attr is allowed as child element of created and deleted as it is also recognizable in Fig. 2. Because the attribute now occurs in the GTXL document independently of its graph element belonging to, it needs to hold a reference. Otherwise the connection between the graph element and its attribute would be lost. Therefore the element attr is linked with an idref to the preserved graph element it is belonging to. In this way the modification of the attribute value can be traced accurately. Note that in version 1.0 of GXL href is not
allowed as an attribute in the element attr. It would be nice if GXL could include this feature in a new version.

To illustrate this by an example, please reconsider the rule LoadTruck in Fig. 4. Imagine the store has an attribute number saving the number of containers kept in the store. An extended LoadTruck rule could express that the store of the RHS keeps one container less than the store of the LHS. Thus, although the store node n4 of the LHS is preserved, the attribute number changes its value. Accordingly in the GTXL document <attr idref="n4" name="number"><string>n</string></attr> is listed in the deleted part and <attr idref="n4" name="number"><string>n-1</string></attr> in the created part of the rule.

5 Summary and Outlook

GTXL is designed to become a standard exchange format for graph transformation systems. The vision of GTXL is manifold. First, it should increase the interoperability between graph transformation tools. E.g. a GTS constructed in AGG [1] could be imported in CheckVML [12] to do model checking on it. An export to GTXL is available in AGG since May ’04. Other tool developers are invited to provide their tools with an export and import functionality. Secondly, software development regarding graph transformation-based tools could become easier using GTXL. The integration of a graph transformation engine is simplified, especially if the implementation language of the engine differs with that of the applying software.

A number of efforts and projects are involved with the design of exchange formats for graphs and transformation-based systems. GraphML [3] for example is another important XML-based graph exchange language. The markup language RuleML [11] is a concrete (XML-based) rule syntax for the Web. In [16] the abstract syntax of RuleML as a MOF [10] model is discussed, with the goal to permit reusability and interchange at a higher level. Another initiative is XMI, XML metadata interchange [18] supporting the interchange of any kind of metadata that can be expressed using the MOF specification. Defining a MOF-metamodel expressing the abstract syntax of a GTS could be another way of handling the exchange of GTSs. The class diagrams related to the gtxl.dtd could be a starting point to define this metamodel.

A number of graph transformation approaches should already be storable in GTXL. But each graph transformation-based tool has its own specialities and might deserve an extension of GTXL. An agenda of not yet supported concepts to be added or extended in order to allow the exchange of GTS between the various existing graph transformation tools is given here. The
elements embedding and transformationUnit are still abstract and have to be further specified. Furthermore, it is necessary to discuss and agree on a clear classification of different GTS-approaches s.t. the use of semantic attributes is efficient. Once a stable version of gtxl.dtd is reached, it should be extended to a schema [20]. XML schemas can express more properties of the data structure than a Document Type Definition.

Additionally, please notice that GTXL does not solve the problem of translating one graph transformation approach into another one e.g. the double-pushout approach into hypergraph replacement. Primarily GTXL will be very convenient to exchange GTSs between tools using the same underlying conceptual approach. In this case a tool2tool cooperation should be relatively straightforward. The exchange, however, of GTSs between tools with a different underlying conceptual approach is more complicated. Assume that a formal method translating GTS1 based on a certain approach into GTS2 based on another approach is available. Note that naturally this translation should be semantics preserving. Since GTS1 can be described in an exchange friendly format like GTXL identifying in particular the underlying approach by means of the semantic attributes, obviously the implementation of its translation into a GTXL document describing GTS2 based on the other approach will be simplified. The Extensible Stylesheet Language XSLT [21] for example is one possible technique to implement the translation of a GTS1 based on approach1 to a GTS2 based on approach2.

References


[20] XML Schema Part 0: Primer, URL: http://www.w3.org/TR/xmlschema-0/

[21] XSL Transformations (XSLT), URL: http://www.w3.org/TR/xslt