Electronic Notes in Theoretical Computer Science 44 No. 4 (2001)
URL: `http://www.elsevier.nl/locate/entcs/volume44.html` 13 pages

Towards Common Exchange Formats for Graphs and Graph Transformation Systems

Gabriele Taentzer

Technische Universität Berlin[1]

Abstract

The development of common exchange formats for graphs and graph transformation systems is an ongoing initiative within the EU Working Group APPLIGRAPH (Applications of Graph Transformation). The author is reporting on the current state of this format discussion. The formats are based on the extensible markup language XML developed to interchange documents of arbitrary types. Graphs are basic structures in various areas of computer science. A common format for graphs supports the interaction of software developers building tools wrt. graph layout, graph algorithms, graph transformation, reengineering, etc. Moreover, within the graph transformation community tool builders can even gain from a common exchange format for graph transformation systems.

1 Introduction

Graphs are a very general data structure used in various fields of computer science and other sciences. To support interoperability between various graph-based tools an ongoing initiative on the development of common exchange formats for graphs and, further on, graph transformation systems has been founded within the EU Working Group APPLIGRAPH. These formats are based on the extensible markup language XML developed to interchange documents of arbitrary types. A first outline of an XML-based exchange format for graphs together with a detailed motivation for such a format has been given in [4].

In September 2000, there was a first meeting (within APPLIGRAPH) on common exchange formats for graphs and graph transformation systems in Paderborn, organized by G. Engels and the author. Participants were G. Busatto, B. Böhlen, R. Depke, R. Heckel, B. Hoffmann, P. Knirsch, M. Matz, M. Minas, P. Rodgers, A. Schürr, G. Valiente, D. Varro, A. Winter and A.

[1] E-mail: {`gabi@cs.tu-berlin.de`}

Zündorf[2]. At this meeting, five different proposals for exchange formats have been made. Most of them were for both, graphs and graph transformation systems. All of them were on the basis of DTDs (Document Type Definitions) which describe the abstract syntax of an XML-document in a grammar-like way. Given a DTD an XML-document can be checked according to validity, i.e. if it is correct wrt. its DTD. During the APPLIGRAPH meeting two common logical models for graphs and graph transformation systems have been discussed. These logical models are slightly more abstract than DTDs and build up on UML class diagrams. Those versions of these logical models which are discussed lateron in Sections 3.1 and 4.1 build up on the meeting results, further discussions with Andy Schürr and a more elaborated version of the GXL model. Please have in mind that the current proposals for common formats are still under consideration and that there will be further meetings to eventually agree on common formats.

This initiative can be seen in connection to others defining common XML-based exchange formats for modelling and specification techniques like UML being XMI [13], Petri nets, rule-based systems, etc. For Petri nets, the PNML (Petri Net Markup Language) has been proposed as a standard interchange format [5].

2 XML Technology

The main purpose of the extensible markup language XML [14] is to provide structured interchange facilities for documents and is meant as a replacement for ASCII. Documents described by XML have tree-like structures and consist of certain entities which contain parsed or unparsed data. Originally XML has been designed to store text documents, nowadays it is more and more used to exchange well-defined data structures between software tools. In this sense, we want to use XML to exchange graph structures and graph transformation systems.

2.1 XML Language Concepts

An XML document has both a logical and a physical structure. Being physically distributed over several entities the logical structure has to form one tree-like structure where all tree elements are properly nested. Each element consists of a start-tag, end-tag, a set of sub-tags (or an empty-tag) and a set of attributes. Tags are enclosed in pointed brackets and the name of each end-tag corresponds to that of its start-tag, but with a leading slash. Additional attributes are depicted inside the start-tag of an element and appear as

[2] Univ. Paderborn: R. Depke, G. Engels, R. Heckel, Techn. Univ. of Berlin: M. Matz, G. Taentzer, Univ. Bremen: G. Busatto, B. Hoffmann, P. Knirsch, RWTH Aachen: B. Böhlen, Univ. Erlangen: M. Minas, Univ. Budapest: D. Varro, Univ. BW München: A. Schürr, Univ. Koblenz: A. Winter, Univ. Kent: P. Rodgers, TU Catalonia: G. Valiente

a list of name-value pairs. Compare the extract of a sample XML document in Section 3.3.

An XML document is *valid* if it has an associated document type definition (DTD) which describes the abstract syntax of the document and if the rest of the document is structured according to this DTD. A DTD provides a grammar for a class of documents. The grammar consists of markup declarations for elements, their attributes, entities or notations. Examples for DTDs are given in Subsections 3.2 and 4.2. It is an ongoing activity in various areas of computer science and further fields to define DTDs for certain classes of document structures. A survey of already defined XML applications and industry initiatives is given at: `http://www.oasis-open.org/cover/xml.html#applications`.

Similarly to DTDs the purpose of XML Schema [15] is to define a class of documents, in fact XML Schema provides functionality which goes above and beyond what is provided by DTDs. The specification of XML Schema is still work in progress. It can be assumed that DTDs will be translated into XML schema after standardization. The following features of XML Schema should be mentioned: unique name spaces for each element, types which can be extended, restricted and redefined as well as abstract elements and types. After standardization of XML Schema we intend to translate the DTDs and logical models developed for graphs and graph transformation systems so far into XML schema. A first approach can be found in [11].

2.2 Supporting Techniques and Tools

Before developing a DTD for a certain document type, a separate design for the document structure may be useful, especially in the case that not text documents but data or object structures should be stored. Here, it is helpful to use a visual technique such as UML class diagrams [10] first and to translate them into DTDs afterwards. Examples for UML class models are given in Figures 1 and 3. The translation can be done mainly automatically.

Two common application programming interfaces (APIs) for XML parsers have been developed: DOM (document object model) [2] defines the logical structure of well-formed XML documents and its manipulation. A DOM model is ideally used as abstract syntax structure of a document serving as a basis for appropriate textual or graphical representations. Another API interesting for XML parsers is SAX (Simple API for XML) [8] to be used for event-based parsing of XML documents. Several XML-based parsers implementing the interfaces SAX and DOM are available now, see e.g. Xerces [12]. XML parsers also support the validation of a document against a certain DTD, similarly there is initial support for XML Schema.

3 GXL: Towards an XML-Format for Exchanging Graphs

Graphs are very general data structures which occur in various fields of computer science. But there are lots of different graph models, dependent on their application context. Considering graphs as they are used for graph layout, graph transformation and reengineering, the exchange format should be able to deal with directed and undirected graphs, hypergraphs, hierarchical graphs, graph types and attributes. In the following discussion, the concrete layout of a graph does not play a role, since there is already the markup language SVG (Scalable Vector Graphics)[9] for this purpose. Furthermore, there is GraphXML [3] another XML based graph interchange format used for graph drawing and visualization. Moreover, we will not go in details concerning graph attributes. Such a general data structure can be attributed with nearly everything. A first outline of GXL (Graph Exchange Language) together with a detailed motivation for such a format has been given in [4].

3.1 The Logical Graph Model

Before developing an XML DTD for graphs we discuss the underlying logical graph model by using UML class diagrams. Compare Figure 1 for the UML class model of graphs. This model is a simplified and slightly modified version of the corresponding model of the GXL model presented in [4]. A GXLDocument contains a set of Graphs where each has a set of *GraphElement*s. A graph element can be a Node, an Edge or a Relation, able to store hyperedges. Edges and relations can both be directed, expressed by their common super class *LocalConnection*. Moreover, partial graphs with dangling edges are allowed, since the multiplicities of associations from and to are allowed to be 0 and 1. Relations have Links storing role names and directions and pointing to graph elements. In this way edges on edges or hyperedges are possible. It was one of our design decisions to distinguish edges from relations which increases the readability of XML documents for simple graphs, although edges may be seen as special relations with two links 'from' and 'to'. Graphs and graph elements are *TypedElement*s meaning that there are type graphs where the graph elements function as types for nodes, edges and relations. It is also possible that a type graph is stored in another GXL document, thus it has to keep a corresponding pointer (Type).

Each graph element is an *AttributedElement* which can have a set of attributes. Each attribute has a name, a type and a value. The value can be a primitive value, a container or a complex value which might be located in another document.

Hierarchical graphs can be stored in several ways: by refinement of graph elements, by attributes or special edges. A complex attribute value could be a graph again such that hierarchical graphs can be described. Here, it is not possible to have edges between graphs in different compound nodes. Another possibility to store hierarchical graphs are special refinement edges

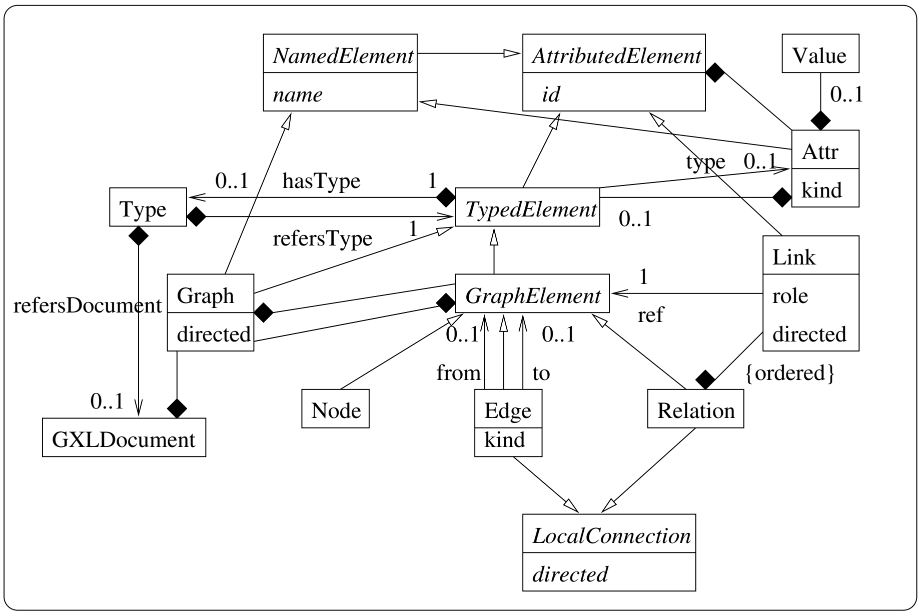

Fig. 1. Logical model for graphs

(to be distinguished by member kind of an Edge). In this case, hierarchical graphs with edges between nodes and compound nodes can be described very generally. Another possibility to store hierarchical graphs is to allow graphs to be graph elements again. This solution is rather restrictive, since it does not allow to refine edges by edge bundles. But it seems to be a good idea to incorporate this possibility into the model to increase the readability of simple hierarchical graphs.

3.2 GXL – Towards a DTD for Graphs

XML documents are trees and storing graphs in trees means linearizing their two-dimensional structure. Such a translation is more or less automatic. In the following, the essential parts of a DTD corresponding to the model in Figure 1 are given. Here, each concrete class of the model results in an element. The class members and associations define the attribute list (ATTLIST) of this element. The associations produce attribute entries pointing to some ID of another element, being an IDREF. The type of such a reference cannot be expressed in a DTD, therefore an additional comment explains which types are expected. Aggregations are translated to subelement relations. Note that in contrast to the logical model subelements can be ordered in a DTD. Abstract classes are not translated to elements, but their members, associations and aggregations are added as attributes and subelements to all elements generated from inheriting classes.

```
<!ELEMENT GXLDocument (Graph*)>

<!ELEMENT Graph (Attr*, (Node | Edge | Relation)*)>
```

```
<!ATTLIST Graph
          id        ID            #REQUIRED
          name      NMTOKEN       #IMPLIED
          type      IDREF         #IMPLIED <!-- Type | Graph -->
          directed (true | false)  "true">

<!ELEMENT Node  (Attr*, Graph* )>
<!ATTLIST Node
          id      ID     #REQUIRED
          type    IDREF  #IMPLIED> <!-- Type -->

<!ELEMENT Edge  (Attr*, Graph*)>
<!ATTLIST Edge
          id       ID      #IMPLIED
          type     IDREF   #IMPLIED <!-- Type -->
          from     IDREF   #IMPLIED <!-- Node | Edge | Relation -->
          to       IDREF   #IMPLIED <!-- Node | Edge | Relation -->
          kind     (refine|normal)       #IMPLIED
          directed (true|false) #IMPLIED>

<!ELEMENT Relation  (Attr*, (Link| Graph)* )>
<!ATTLIST Relation
          id        ID       #IMPLIED
          type      IDREF    #IMPLIED <!-- Type -->
          directed  (true|false) #IMPLIED>

<!ELEMENT Link (Attr*)>
<!ATTLIST Link
          ref      IDREF   #REQUIRED<!-- Node | Edge | Relation -->
          role     NMTOKEN #IMPLIED
          directed (true|false) #IMPLIED>

<!ELEMENT Attr  (Attr*, Value? )>
<!ATTLIST Attr
          name   NMTOKEN  #REQUIRED
          type   IDREF    #IMPLIED <!-- Type -->
          kind   NMTOKEN  #IMPLIED>
```

3.3 A Sample GXL Graph

In Figure 2, a sample graph is given which contains nearly all structural features a graph can have. Node A is refined to the graph part drawn inside of this node. There are relations (hyperedges) f and g where one link of f points to g. Edge a is partial, it does not have a target. Edges c and d are parallel edges between the same nodes C and D and edge e is a loop. Edge b runs between a simple and a compound node. A section of a GXL document

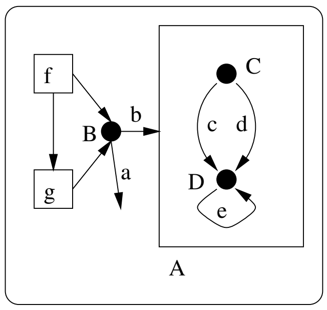

Fig. 2. A sample graph

corresponding to the graph in Figure 2 and to the DTD given in the previous subsection is shown below.

```
<Graph id="G">
   <Node id="A"> </Node>
     <Graph id = "sub-A">
      <Node id="C"> </Node>
      <Node id="D"> </Node>
      <Edge id="c" from="C" to="D"> </Edge>
      <Edge id="d" from="C" to="D"> </Edge>
      <Edge id="e" from="D" to="D"> </Edge>
     </Graph>
   </Node>
   <Node id="B"> </Node>
   <Edge id="a" from="B"> </Edge>
   <Edge id="b" from="B" to="A"> </Edge>
   <Relation id="f">
      <Link ref="B"> </Link>
      <Link ref="g"> </Link>
   </Relation>
   <Relation id="g">
      <Link ref="B"> </Link>
   </Relation>
</Graph>
```

4 GTXL: Towards an XML-Format for Exchanging Graph Transformation Systems

Building up on GXL as exchange format for graphs we consider the exchange of graph transformation systems now. Such a format, called GTXL in the following, is of interest within the graph transformation community to interchange not only graphs but also graph rules, control structures, etc. of graph transformation systems. The graph transformation tools available so far put their main emphasis on different aspects. For graph transformation convenient interpretation, analysis of rules and transformations, or code generation is supported, but not by one and the same tool. A common exchange format supported by the graph transformation tools could help in cooperative work with several tools.

4.1 The Logical Model for Graph Transformation Systems

Analogously to GXL also for GTXL a logical model is created first. Compare Figure 3. A GTS (Graph Transformation System) consists of a set of Graphs, a set of Operations and maybe a Type (graph). If the set of graphs has cardinality 1, a graph grammar is given. The storage of further graphs might be useful to keep intermediate results or to define possible results.

The operations can be grouped in an OpGroup which is a certain *Control-Structure* being again an operation, e.g. rules can be grouped in transformation units [1] which are operations themselves. Each operation can have a list of Parameters where each one contains a name and a ptype. A special operation is a Rule which consists of two RuleGraphs and a Mapping. A RuleGraph contains not only a Graph, but also a set of *Condition*s which may be used as pre- or postconditions. These conditions can be AttrConditions containing boolean expressions on attributes or GraphConditions. Each GraphCondition contains a Mapping and a RuleGraph again such that GraphConditions may be nested. Each Mapping consists of a set of MapElements running between two *GraphElement*s. A rule may have an additional Embedding part where the embedding of the right-hand side rule graph into the context graph is specified.

Considering various graph transformation approaches, they differ heavily in the definition of a rule. It is our intention to provide a syntax for rules which is general enough to capture those various approaches. What is common to all these approaches is the specification of a rule by three graph parts: one to be deleted, one to be preserved and one to be created. These are described by two rule graphs and a mapping in between. All those parts of the first graph which are not mapped are deleted. All mapped parts are preserved and all parts of the second rule graph to which there isn't a mapping are created.

Several graph transformation approaches support negative application conditions. This kind of conditions can be describe as graph condition with a mapping from the left-hand side of the corresponding rule to another graph containing additional graph parts expressing what is not allowed. Moreover, the logical model allows to nest graph conditions such that implications are possible.

It's up to the reader to compare this first version of a logical model for graph transformation systems more closely with the main graph transformation approaches. Several concepts are not yet refined, i.e. ControlStructure, Embedding and Expression. The refinement of these concepts will be one of our next tasks.

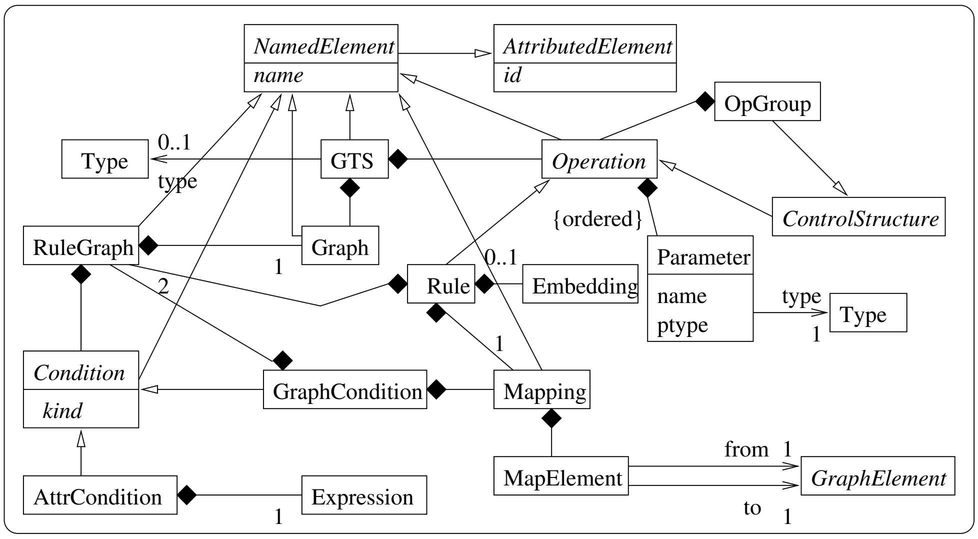

Fig. 3. Logical model for graph transformation systems

4.2 GTXL – Towards a DTD for Graph Transformation Systems

The generation of a DTD from a given logical model has been shortly described in Section 3.2. The following extract of the DTD for GTXL is generated in the same way.

```
<!ELEMENT GTS (Attr*,(Graph | Rule | OpGroup)*)>
<!ATTLIST GTS
          id   ID      #REQUIRED
          name NMTOKEN #IMPLIED
          type IDREF   #IMPLIED><!-- Type -->

<!ELEMENT OpGroup EMPTY>

<!ELEMENT Rule (RuleGraph, RuleGraph, Mapping, Parameter*, Embedding?,
                                                               Attr*)>
<!ATTLIST Rule
          id    ID      #REQUIRED
          name  NMTOKEN #IMPLIED>

<!ELEMENT Mapping (Attr*, MapElem*)>
<!ATTLIST Mapping
          id     ID       #REQUIRED
          name   NMTOKEN  #IMPLIED>

<!ELEMENT MapElem EMPTY>
<!ATTLIST MapElem
          from   IDREF #REQUIRED  <!-- Node | Edge | Relation -->
          to     IDREF #REQUIRED> <!-- Node | Edge | Relation -->

<!ELEMENT RuleGraph (Graph, (AttrCondition | GraphCondition)*)>
```

```
<!ATTLIST RuleGraph
          id     ID      #REQUIRED
          name   NMTOKEN #IMPLIED>

<!ELEMENT GraphCondition (Attr*, RuleGraph, Mapping)>
<!ATTLIST GraphCondition
          id     ID       #REQUIRED
          name   NMTOKEN  #IMPLIED
          kind   NMTOKEN  #IMPLIED>

<!ELEMENT AttrCondition (Expression)>
<!ATTLIST AttrCondition
          id     ID       #REQUIRED
          name   NMTOKEN  #IMPLIED
          kind   NMTOKEN  #IMPLIED>

<!ELEMENT Expression (#PCDATA)>

<!ELEMENT Embedding (#PCDATA)>

<!ELEMENT Parameter (EMPTY)>
<!ATTLIST Parameter
          name   NMTOKEN  #REQUIRED
          type   IDREF    #REQUIRED  <!-- Type | Node |
                                          Edge | Relation | Attr -->
          ptype  (in|out|inout) #REQUIRED>
```

GTXL is intended to function as an exchange format for all the main graph transformation approaches such as node replacement, hyperedge replacement, single and double-pushout approaches, algorithmic approaches, etc. as presented in [6]. The main concept of graph transformation systems is that of rules. Though it would beinteresting to compare the format definition of GTXL with the Rule Markup Language (RuleML) [7] developed in the area of artificial intelligence.

4.3 A Sample Graph Transformation System

In Figure 4, a very simple graph transformation system is given as example. It is a graph grammar with one start graph and a rule. The rule creates a new edge with a new target node if the source node is connected to another node by an hyperegde. The hyperedge and the further node are deleted. Furthermore, the rule contains a negative application condition which prohibits the application of the rule if a loop edge is attached to the source node.

A GTXL section corresponding to the given graph grammar and the DTD given in the previous subsection is shown below.

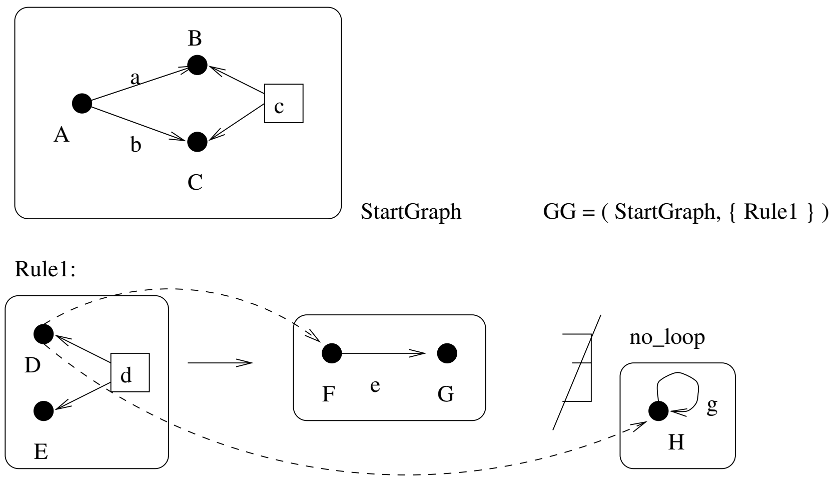

Fig. 4. Sample graph transformation system

```
<GTS id="gg0" name="GG">
  <Graph id="g" name="StartGraph" directed="true">
    <Node id="A"> </Node>
    <Node id="B"> </Node>
    <Node id="C"> </Node>
    <Edge id="a" from="A" to="B"></Edge>
    <Edge id="b" from="A" to="C"></Edge>
    <Relation id="c">
      <Link ref="B"> </Link>
      <Link ref="C"> </Link>
    </Relation>
  </Graph>
  <Rule id="r" name="Rule1">
    <RuleGraph id="lhs">
      <Graph id="l" directed="true">
        <Node id="D"> </Node>
        <Node id="E"> </Node>
        <Relation id="d">
          <Link ref="D"> </Link>
          <Link ref="E"> </Link>
        </Relation>
      </Graph>
      <GraphCondition id="gc"  name="no_loop">
        <RuleGraph id="nac">
          <Graph id="n" directed="true">
            <Node id="H"> </Node>
            <Edge id="g" from="H" to="H"></Edge>
          </Graph>
        </RuleGraph>
        <Mapping id="m">
```

```
          <MapElem from="D" to="H"></MapElem>
        </Mapping>
       </GraphCondition>
    </RuleGraph>
    <RuleGraph id="rhs">
      <Graph id="r" directed="true">
        <Node id="F"> </Node>
        <Node id="G"> </Node>
        <Edge id="e" from="F" to="G"></Edge>
      </Graph>
    </RuleGraph>
    <Mapping id="m">
      <MapElem from="D" to="F"></MapElem>
    </Mapping>
  </Rule>
</GTS>
```

5 Conclusion

Finding a common XML-based exchange format for graphs and graph transformation systems is an ongoing initiative within the EU Working Group APPLIGRAPH. Currently, common logical models are discussed and we plan to fix a kernel of the format in March 2001. Preliminary versions of the formats are already implemented in several graph-based tools to gain experience with the XML technology. Moreover, the new XML-based formats can be used as new storing formats for graphs and graph transformation systems which are human-readable. It is our hope that the common formats we decide on soon will be supported by various graph-based tools and increase their interoperability heavily.

References

[1] M. Andries, G. Engels, A. Habel, B. Hoffmann, H.-J. Kreowski, S. Kuske, D. Plump, A. Schürr, and G. Taentzer. Graph transformation for specification and programming. *Science of Computer Programming*, 34:1–54, 1999.

[2] Document Object Model (DOM) Level 2 Core Specification. available at: `http://www.w3.org/TR/DOM-Level-2`.

[3] GraphXML. available at: `http://www.cwi.nl/InfoVisu/GraphXML`.

[4] R. Holt, A. Winter, and A. Schürr. GXL: Towards a Standard Exchange Format. In *Proc. 7th Working Conference on Reverse Engineering (WCRE 2000)*, pages 162 – 171, Los Alamitos, 2000. IEEE Computer Society.

[5] M. Jüngel, E. Kindler, and M. Weber. The Petri Net Markup Language. In S. Philippi, editor, *Algorithmen und Werkzeuge für Petrinetze (AWPN), Koblenz, June 2000*, 2000. available at: `http://www.informatik.hu-berlin.de/top/pnml`.

[6] G. Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformations, Vol. 1: Foundations.* World Scientific, 1997.

[7] Rule Markup Language (RuleML). available at: `http://www.dfki.uni-kl.de/ruleml`.

[8] SAX 2.0: The Simple API for XML. available at: `http://www.megginson.com/SAX`.

[9] Scalable Vector Graphics. available at: `http://www.w3.org/TR/SVG`.

[10] *Unified Modeling Language – version 1.3*, 2000. Available at `http://www.omg.org/uml`.

[11] D. Varro and A. Pataricza. An XML Schema Description of Graph Transformation Systems. Technical report, Budapest University of Technology and Economics, 2000.

[12] Xerces. available at: `http://xml.apache.org/`.

[13] XML Metadata Interchange. available at: `http://www.oasis-open.org/cover/xmi.html`.

[14] Extensible Markup Language (XML). available at: `http://www.w3c.org/xml`.

[15] XML Schema Part 0: Primer. available at: `http://www.w3.org/TR/xmlschema-0`.