GraphXML — An XML-Based Graph Description Format

Ivan Herman and M. Scott Marshall

Centre for Mathematics and Computer Sciences (CWI)
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands
Email: {I.Herman, M.S.Marshall}@cwi.nl

Abstract. GraphXML is a graph description language in XML that can be used as an interchange format for graph drawing and visualization packages. The generality and rich features of XML make it possible to define an interchange format that not only supports the pure, mathematical description of a graph, but also the needs of information visualization applications that use graph–based data structures.

1 Introduction

Most graph drawing systems could benefit from a textual description language to input and, possibly, to output graphs in a human readable form. Ideally, such a description language could also serve as a standardized format for data exchange between systems, enabling information exchange. Defining such a description language is not, by itself, a particularly difficult task: after all, a graph is simply a collection of nodes and edges. Several formats have been proposed in the past such as GML[1] and WebDot's DOT format[2], and the formats used by Rigi[3], LEDA[4], and GDS[5]. None of these are universally supported and they are usually bound to specific systems.

An important development of recent years, which also influences the choice of interchange formats, is the synergy between graph drawing techniques and information visualization. Information visualization has become a well-known research area, with important industrial and scientific applications. Graph drawing techniques play a prominent role in information visualization because the data structures to be visualized can often be described as graphs. The demands of information visualization pose new challenges with respect to graph description formats. It is necessary to include features in a graph description language that are not directly relevant to pure graph drawing. For example, the graph description should be able to include application dependent data[1], either embedded in the description itself or externally referenced (e.g., network statistics for a Web visualizer, genetic data for consensus trees as used in evolutionary research, database references for the result of a database search, etc.). Furthermore, it should be possible to describe composite structures such as nested

[1] Although GML, for example, is a capable description language for graph drawing purposes and includes provisions for extension, the mechanism for associating external data with a graph element is not well defined.

J. Marks (Ed.): GD 2000, LNCS 1984, pp. 52-62, 2001.

hierarchies and clusters. Another less obvious feature is to support the description of the evolution of graphs in time for applications such as animation.

As part of a larger project in graph visualization, we needed a graph description language. Although, initially, we looked for an existing standard, like the ones cited above, none could fully support the needs of information visualization. Consequently, we decided to develop our own format with particular attention to the requirements of information visualization. The result is GraphXML, a graph description language based on XML. Description of this language is the subject of this paper. We hope that this format will enter widespread use. In our view, this would be beneficial for the graph visualization community.

2 Why XML?

XML is a language specification developed by the World Wide Web Consortium that has received significant attention in the last few years[2]. The future evolution of the Web, both in the traditional Internet domain and in mobile communications, is based on XML. There are several reasons for choosing XML as the basis for a graph description format:

- XML defines clear syntactic rules for specifying a "language" for a particular application. An application-specific language is defined in a file called a Document Type Definition (DTD). The end–user also has the option of adding extensions to the language specified in the DTD.
- XML is used by many different applications to define data formats including those for databases, chemical compound definitions, e-commerce, mobile devices, and schematic graphics. A graph interchange format based on XML has a greater chance of being accepted by other application communities.
- There are a number of XML–based specifications that are being defined by communities, both within and outside the World Wide Web Consortium. GraphXML can take advantage of some of these existing standards. See Section 0 for examples.
- Software tools are emerging, which are either based on XML or work with XML. For example, there are a number of both commercial and public domain XML editors. These tools can be very helpful in managing graph description files that are based on GraphXML.
- Several XML parser tools are freely available in Java, C, and C++. A full-featured parser that provides error management and syntax checking can easily be generated using these tools. The main task is to define the semantic interpretation specific to the application (in our case, GraphXML).

[2] There are hundreds of books on XML, so rather than pick a specific reference we refer to the original specification[6], which is available on-line.

In what follows, an overview of the main features of GraphXML is given. A more complete description of GraphXML, including the exact specification, is available in [7].

3 Graph Structures in GraphXML

3.1 A Simple Example

The following code segment shows the simplest possible use of GraphXML. It describes a graph with two nodes and a simple edge:

```
1    <?xml version="1.0"?>
2    <!DOCTYPE GraphXML SYSTEM "file:GraphXML.dtd">
3    <GraphXML>
4      <graph>
5        <node name="first"/>
6        <node name="second"/>
7        <edge source="first" target="second"/>
8      </graph>
9    </GraphXML>
```

This example shows the basic style of a graph description in GraphXML. It resembles the way HTML documents are written, albeit using different tags. This example contains all the elements that are necessary to describe a purely mathematical graph.

The first line is required in all XML files. The second line specifies that this is an XML application based on the GraphXML DTD contained in the file `GraphXML.dtd`[1]. Finally, the third and the last lines enclose the real content of the files. The real content begins with line number 4, which defines a full graph. We delineate graph definitions with the `<graph>` tag so that a file can contain several graph definitions. The body of the graph description is straightforward: two nodes and a connecting edge are defined.

Attributes can also be defined for each of the elements: these are key–value pairs. The GraphXML DTD defines the set of allowable attributes for each element and a validating parser will to check those. It is partly through those attributes that additional information about nodes, edges, or graphs can be conveyed to the application. For example, the `<graph>` tag can use keys such as `version`, `vendor`, `preferredLayout`, `isPlanar`, `isDirected`, `isAcyclic`, or `isForest`.

3.2 Application-Dependent Data

Application data can be added to different levels of the graph description through a series of additional elements. These elements are meant to represent the different types of application or domain-dependent data that can be associated with a graph, a node, or an edge. GraphXML defines the following application data:

[1] This line specifies that the DTD is on the local file system but could be replaced by a URL specification. In this way, it is possible to make the DTD publicly available on the World Wide Web.

- **Labels** (<label> tag): whereas the name attribute is used for unique identification, the label can contain any kind of text and does not have to be unique. Applications can use these to label nodes and edges.
- **Data** (<data> tag): domain-dependent data represented by a node, edge, or even the full graph[1].
- **Data references** (<dataref> tag): data that is referenced externally rather than embedded in the graph description file.

The format of external references (within the <dataref> tag) follows the specification of a separate document of the World Wide Web consortium, called XLink[8]. For our purposes, the virtually identical URL format used in HTML will suffice for external references.

The following example describes the same graph as in the previous section, except that the first node has associated data[2].

```
1           <node name="first">
2            <label>Project Home page</label>
3             <data> CWI Information Visualization project</data>
4             <dataref>
5              <ref xlink:role="Lead"  xlink:href="http://…/~ivan"/>
6              <refxlink:role="Descr"xlink:href="http://…/InfoVisu"/>
7             </dataref>
8            </node>
9           <node name="second"/>
10          <edge source="first" target="second"/>
```

Note the use of the xlink:role attribute in the example (see lines 0 and 0), which can be used to describe what the exact role of the link is. This can provide a useful indication to the application.

3.3 Hierarchical Graphs

All the examples mentioned until now refer to a single graph. However, information visualization applications are often used on very large graphs, and one of the ways of handling the size problem is to define the information in terms of a hierarchy of graphs, or clusters, instead of one single graph. What this means is that the nodes of one graph can refer to other graphs, these can refer to yet another graph, and so on. Powerful techniques exist to hierarchically cluster graphs and to visualize the hierarchies (see the survey of Herman *et. al.*[9]). GraphXML offers a way to describe such graph hierarchies. Consider the following example:

```
1           <graph id="L-1">
2             <node name="first"/>
3             <node name="second"/>
```

[1] Although XML describes everything in terms of strings, it has its own formalism, called entities and notations, which can be used to include binary data, too. However, using external references, through the <dataref> tag, may be more appropriate for this.

[2] From now on, we are omitting the header part from the examples to save space.

```
4             <edge source="first" target="second"/>
5           </graph>
6           <graph id="L-2">
7             <node name="third"/>
8             <node name="fourth"/>
9             <edge source="third " target="fourth"/>
10          </graph>
11          <graph id="levelTwo">
12          <node isMetanode="true" name="cluster1" xlink:href="#L-1"/>
13          <node isMetanode="true" name="cluster2" xlink:href="#L-2"/>
14          <edge source="cluster1" target="cluster2"/>
15          </graph>
```

The example shows the tools introduced in GraphXML to describe graph hierarchies. A GraphXML document can contain several graph descriptions. Each graph description can use a (unique) identifier, using the `id` key. A "meta" node in a higher level in the hierarchy uses this identifier to "`link`" to another graph, using the `xlink:href` attribute key (see line numbers 0 and 0). The `isMetanode` attribute is used to unambiguously identify a node that refers to another graph. Using these definitions, this example describes a two level graph with two nodes and one connecting edge, where each node represents another graph. The full format of the `xlink:href` value is: `URL#identifier` where the identifier refers to a graph identifier *within* the document referred to by the URL. If the target is in the same document as the source, the URL part can be left out (as in the example).

This simple adaptation of HTML introduces an powerful feature to GraphXML. It is possible to define a hierarchy of graphs, consisting of graphs that that are located in another file, or possibly in another Internet location than the hierarchy description itself. Applications that make use of this capability can create their own hierarchical or clustered views of public datasets.[1]

3.4 Dynamic Graphs

If a graph visualization system is used interactively, the system may be asked to store the history of the user's actions in some form of journaling. What this means is that an interchange format should be able to describe not only the initial graph, but also any editing steps that have changed the structure or the attributes of the graph. This is the reason for the use of `<edit>` tags in GraphXML.

The edit sections in a GraphXML document are syntactically similar to graph specification, except for the use of the `<edit>` tag instead of `<graph>`. Furthermore, the `<edit>` tag has a required attribute key `action`, whose value can be `remove` or `replace`. Here is an example:

[1] Note that WebDot's DOT format[2] includes facilities to describe clusters, but the format is limited to subgraphs defined in the same file.

```
1          <graph version="1.0" vendor="cwi" id="theGraph">
2            <node name="first">
3              <label>A label to display for this node</label>
4              <dataref> <ref xlink:href="BigIcon.gif"/> </dataref>
5            </node>
6            <node name="second"/>
7            <edge name="thisEdge" source="first" target="second">
8          <dataref> <ref xlink:href="ExternalData.bmp"/> </dataref>
9            </edge>
10         </graph>
11         <edit action="replace" xlink:href="#theGraph">
12           <node name="first">
13           <label>Another label</label>
14           <dataref> <ref xlink:href="anotherImage.gif"/> </dataref>
15           </node>
16           <edge name="thisEdge" source="first" target="second"/>
17         </edit>
```

The semantics of the editing element (line 0) is based on identifying the element in the edit block and in the original graph. This controls what is being edited. The value of the `action` attribute determines what happens: the corresponding element will either be removed or replaced by the content in the `<edit>`. The semantics of matching elements is more involved (see the full description[7] for details).

The result of carrying out the editing action in the above example can be represented by the following GraphXML description:

```
1          <graph version="1.0" vendor="cwi" id="theGraph">
2           <node name="first">
3            <label> Another label </label>
4           <dataref> <link xlink:href="anotherImage.gif"/> </dataref>
5            </node>
6            <node name="second"/>
7            <edge name="thisEdge" source="first" target="second"/>
8          </graph>
```

Note the disappearance of the data references in the edge (line 0 in the previous example). This is because the editing action has replaced those with their "empty" counterpart from within the editing element (line 0 in the previous example).

If, in the same example, the `action` attribute were set to `remove`, the result would be as follows:

```
1  <graph version="1.0" vendor="cwi" id="theGraph">
2   <node name="first"> </node>
3   <node name="second"/>
4   <edge name="thisEdge" source="first" target="second">
5   </edge>
6  </graph>
```

The `xlink:href` attribute used by the edit element has the same syntax and semantics as described for hierarchical graph descriptions. In other words, an edit element can refer to a graph in another file or even another Internet location.

As an additional tool for editing, GraphXML also defines the `<edit-bundle>` tag, which is simply a sequence of edit tags:

```
1 <edit-bundle>
2  <edit …> … </edit>
3  <edit …> … </edit>
4 </edit-bundle>
```

This simple grouping of editing elements can be useful if the user wants to animate the result of editing, but doesn't want to display each individual editing step. Using this bundling mechanism, the granularity of animation can be controlled by the creator of the GraphXML file.

4 Storing Geometry

Section 0 describes only structural elements: nodes, edges, and hierarchies. Visualization systems have to layout the graph before presenting it to the user. GraphXML tags for this purpose are described in the next section.

4.1 Positions and Size

The position of a node can be described by adding the <position> tag as a child to <node>:

```
1    <position x="0.0" y="0.0"/>
```

The size of the node can also be described with the <size> tag:

```
1    <size width="3.0" height="5.0"/>
```

This tag can be especially important for layout algorithms that take the node size into account when laying out the graph.

The <size> tag can also be used as a direct child of <graph>. It then denotes the size, or bounding box, of the full graph. Applications can benefit from such information because it allows them to allocate the necessary area on the screen and coordinate transformations in advance.

4.2 Edge Geometry

Edges differ from nodes insofar as a sequence of coordinates may be necessary. This is achieved through the <path> tag:

```
1 <path type="polyline">
2    <position x="0.0" y="0.0"/>
3    <position x="0.1" y="0.0"/>
4    <position x="0.1" y="0.1"/>
5 </path>
```

The <path> tag contains a sequence of control points. The type attribute can take the value of polyline, arc, or spline, depending on whether the edge is to be drawn as a polyline or a spline curve. In the case of a spline, the positions indicate the spline control points.

4.3 Geometry for Hierarchical Graphs

The geometry definition described earlier is insufficient for the description of hierarchical graphs: the same graph might be included in more than one place in the higher-level graph and the geometry must be adapted to the metanode's position. The solution is to use the <transform> tag, which is a child of <node>. This element describes the transformation to be applied to each coordinate value in the referenced graph. For example, the following code fragment:

```
1  <node name="SecondOrder" isMetanode="true" xlink:href="#basic">
2      <transform matrix="1.0 0.0 0.5 0.0 1.0 0.5"/>
3  </node>
```

translates all the referred nodes and points to the (0.5,0.5) point. The <transform> element contains 6 numbers to describe a 2×3 matrix. See [10] for details on how these transformation matrices are used in computer graphics.[1]

5 Visual Properties

In addition to layout, the appearance of a graph is determined by visual properties, such as line width, colour of the components, icons replacing nodes, etc. In GraphXML, the user can control these properties through the <style> tag. A style can include the tags <line> or <fill>. In the case of a node, the line tag controls the border of the symbol drawn for the node, whereas the fill tag controls the interior. For example, the block:

```
1 <node name="first">
2  <style>
3    <line linestyle="dashed" linewidth="2" colour="red"/>
4    <fill fillstyle="solid" colour="blue"/>
5  </style>
6 </node>
7 <edge source="first" target="second">
8   <style>
9     <line linestyle="solid" linewidth="1" colour="cyan"/>
10    <fill fillstyle="none"/>
11   </style>
12 </edge>
```

defines a node symbol to have a red dashed boundary drawn with a line width of 2, and filled with solid blue[2]. The edge should be drawn in cyan without filling the interiors (in case the edge is drawn as a polygon).

The fill element can also refer to an image file instead of specifying the colour and the fill style. This instructs the application to use the image as an icon to display the node. For example, line 0 could be replaced by:

```
      <fill xlink:href="http://www.some.site/imagefile.gif"/>
```

[1] All positions, as well as the transformations, can also be extended to 3D.

[2] Note that this specification does not specify the exact glyph to be drawn by the application. This is either left to the implementer of the visualization system, or specified via a more sophisticated control tag, called <implementation>. See [7] for further details.

The mechanism described so far would lead to repeated visual control tags, greatly increasing the size of the graph file. It might also become cumbersome to adapt a graph file to a new environment with other visual characteristics. To solve these problems, an inheritance mechanism for visual properties is available. The goal is to provide a general control mechanism that allows for easy adaptation. A style element can be added on the graph level, to control the overall appearance of the graph:

```
1 <graph>
2  <style>
3   <line tag="node" linestyle="dashed" linewidth="2" colour="red"/>
4   <fill tag="node" fillstyle="solid" colour="blue"/>
5   <line tag="edge" linestyle="solid" linewidth="1" colour="cyan"/>
6   <fill tag="edge" fillstyle="none"/>
7  </style>
8  …
```

This results in the same visual effect as before, except that the visual properties are valid for *all* nodes and edges in the graph (note the use of the `tag` attribute in the line and fill elements to differentiate between nodes and edges).

Nodes and edges can also use the `class` attribute to categorize elements with common visual properties. Using this additional identification, finer control over the visual attributes can be achieved by applying a specific visual attribute to a class of nodes or edges. For example, in following block of code:

```
1 <graph>
2  <style>
3   <line tag="node" linestyle="dashed" linewidth="2" colour="red"/>
4   <fill tag="node" fillstyle="solid" colour="blue"/>
5   <fill tag="node" colour="green" class="special"/>
6  </style>
7  …
8  <node name="first"/>
9  <node name="second" class="special"/>
10  …
11  <node name="nth" class="special"/>
12  …
```

the node `first` will be displayed the same way as before; however the nodes `second` and `nth` will become green instead of red. This is because line 0 specifies that "all nodes of class 'special' should be filled in green".

Metanodes require a special style control facility to affect the style of all included elements. The reader should refer to [7] for details.

Using the entity mechanism of XML, it is possible to include an XML file within another. This is particularly handy when controlling styles: it is possible to collect all the style elements into a separate file and include it in a graph specification. If a change is made to the style file, it will automatically affect the visual properties of all the graph description files that reference it.

6 User Extensions

Although the specification of GraphXML includes rich facilities for the association of data with nodes and edges, it is not possible to predict all the possible attributes an application might want to add to an element. For example, if the application is for web visualization, the user might want to associate a MIME type with a node. In other

words, the application would need to have its own, <mime> tag for each node, in addition to the tags defined by the GraphXML DTD.

Such extensions are possible using GraphXML. This is how an extension for a mime tag can be added to a graph:

```
1 <!DOCTYPE GraphXML SYSTEM "file:GraphXML.dtd" [
2 <!ENTITY % nodeExtensions "|mime">
3 <!ELEMENT mime EMPTY>
4 <!ATTLIST mime
5    type        CDATA #REQUIRED
6    application CDATA #IMPLIED
7 >
8 ]>
9 <GraphXML>
10   <graph>
11    <node name="first">
12      <label>Project Home page</label>
13      <dataref> <ref xlink:href="Description.pdf"/> </dataref>
14      <mime type="application/pdf" application="Adobe Acrobat"/>
15    </node>
16    …
```

The file contains what is called an "internal DTD" (lines 0–0), which extends the elements that can be included in a node definition. The definition states that a <mime> element can be added as the child of a node, that this is an element that contains only attributes (i.e. no sub–elements), and that the attributes can be type and application (the first being compulsory, the second optional).

The XML syntax is a bit cryptic. However, the end–user does not necessarily have to include such internal DTD's into all GraphXML files. Instead, using standard XML mechanisms, it is possible to define a *separate* DTD containing the application-specific extensions (and a reference to the GraphXML.dtd, of course). Using such extension, the header part becomes simply:

```
1 <!DOCTYPE GraphXML SYSTEM "file: WebVisualizer.dtd ">
2 <GraphXML>
3  <graph>
4    …
```

In other words, the intricacies of the XML DTD syntax remain invisible to most users. Furthermore, because the extension is made through a standard XML mechanism, the basic GraphXML parser remains unchanged. Only the application–dependent part has to be adapted for the new extensions. Herman and Marshall [7] describes application-specific DTD's in more detail.

7 Future Developments

As stated REFearlier, one of the advantages of using XML is that the future evolution of XML-based specifications can be used to develop new tools for GraphXML. Here are just two examples:

- An upcoming specification is the RDF Schemas[11], which will replace DTD's in future. Schemas also include various data types with possible constraints on the values. Schema based parsers will be able to make on-the-fly checks on the input

data (e.g., coordinates should be numbers), relieving the GraphXML parser and applications from having to perform these checks themselves.

- W3C is currently developing a standard for graphics on the Web, called SVG[12]. It will be possible to define simple tools to transform GraphXML specifications into SVG[1]. This means that the result of graph drawing systems can be published on the Web in the form of vector based graphics, rather than screen dumps, yielding better quality and smaller bandwidth requirements.

8 Public Availability

The full description of GraphXML, as well as the GraphXML DTD, is publicly available at the URL: http://www.cwi.nl/InfoVisu/GraphXML. The parser is also available as a collection of Java 1.2 classes, which can be embedded into an application. The implementation uses publicly available XML parsers in Java. See the full description at the above URL for details.

References

1. M. Himsolt, *GML - Graph Modelling Language,* http://infosun.fmi.uni-passau.de/Graphlet/GML/, 1997.
2. S. C. North, *Dot Abstract Graph Description Format,* http://www.research.att.com/~north/cgi-bin/webdot.cgi/dot.txt.
3. K. Wong, *Rigi Users' Manual,* http://www.rigi.csc.uvic.ca/rigi/manual/user.html, 1996.
4. S. Thiel, *LEDA Graph Input/Output Format,* http://www.mpi-sb.mpg.de/LEDA/information/graph_io_format/graph_io_format.html, 1999.
5. *GDS Supported File Formats,* http://loki.cs.brown.edu:8081/geomNet/gds/formats.shtml, 2000.
6. "Extensible Markup Language (XML) 1.0", World Wide Web Consortium, (eds. T. Bray, J. Paoli, C.M. Sperberg–McQueen), Recommendation February 1998, http://www.w3.org/TR/REC-xml.
7. I. Herman and M. S. Marshall, "GraphXML — an XML Based Graph Interchange Format", Centre for Mathematics and Computer Sciences, Amsterdam INS-R0009, 2000, http://www.cwi.nl/InfoVisu/GraphXML/GraphXML.pdf.
8. "XML Linking Language (XLink)", World Wide Web Consortium, (eds. S. DeRose, D. Orchard, B. Trafford), Working Draft July 1999, http://www.w3.org/TR/WD-xlink.
9. I. Herman, M. S. Marshall, and G. Melançon, "Graph Visualisation and Navigation in Information Visualisation: A Survey", *IEEE Transactions on Visualization and Computer Graphics*, vol. 6, pp. 24-43, 2000.
10. J. D. Foley, A. v. Dam, and S. K. Feiner, *Computer Graphics: Principles and Practice* Reading, Addison–Wesley, 1990.
11. "Resource Description Framework (RDF) Schema Specification 1.0", World Wide Web Consortium, (eds. D. Brivkley and R.V. Guha), Candidate Recommendation March 2000, http://www.w3.org/TR/2000/CR-rdf-schema-20000327/.
12. "Scalable Vector Graphics (SVG) 1.0 Specification", World Wide Web Consortium, (eds. D. Ferraiolo et al.), Working Draft March 2000, http://www.w3.org/Graphics/SVG/Overview.htm8.

[1] Standard XML-based tools can perform such a transformation. Also, a graph vizualization application that reads GraphXML and saves to SVG format can be found at http://www.cwi.nl/InfoVisu/GVF/ .