GraphEx: An Improved Graph Translation Service

Stina Bridgeman

Computer Science Department, Colgate University, Hamilton, NY 13346 USA
sbridgeman@mail.colgate.edu

Abstract. The Internet-based translation service GraphEx automatically converting between different graph formats, making it easier for users to take advantage of the full variety of graph drawing tools. GraphEx builds on the prototype translation component of the Graph Drawing Server [2], improving the handling of information mismatch problems and adding support for user-defined formats.

1 Introduction

The development of a wide variety of graph drawing tools and libraries has led to the creation of many different formats for representing graph-structured information. The simplest formats contain only a list of edges to describe the combinatorial structure of the graph, while the most sophisticated allow labels, geometry, and other attributes to be associated with the graph structure. The variety of graph formats can make it difficult for people to take full advantage of the available graph drawing technology because of the effort required to translate existing graph data to the required format(s).

One solution is to develop a common graph interchange format flexible enough to support application-specific data. Efforts in this direction include GML [4], GraphXML [3], GXL [5], and GraphML [1]. These formats specify a core representation for the combinatorial structure of the graph and a framework for adding arbitrary attributes. Well-behaved applications are expected to quietly ignore (and possibly preserve) unknown attributes.

However, problems remain: there are multiple competing interchange formats, applications already in existence still use their own specialized formats, and graph-structured data arises in many domains and a standard format in one domain may be unfamiliar in another. Furthermore, implementors of a new tool may define their own format because the existing formats do not support the features they need or are too complex to parse, or because they are not aware of a suitable format. As a result, there is a need for a graph translation service which can automatically convert between different graph formats.

GraphEx (for *G*raph *Ex*change) improves on the graph translation component of the Graph Drawing Server [2] by reducing unnecessary loss of information and by supporting user-defined formats. A key component in reducing information loss is the incorporation of a merging step to restore attributes lost as a result of the format required by the desired drawing algorithm.

G. Liotta (Ed.): GD 2003, LNCS 2912, pp. 307–313, 2004.

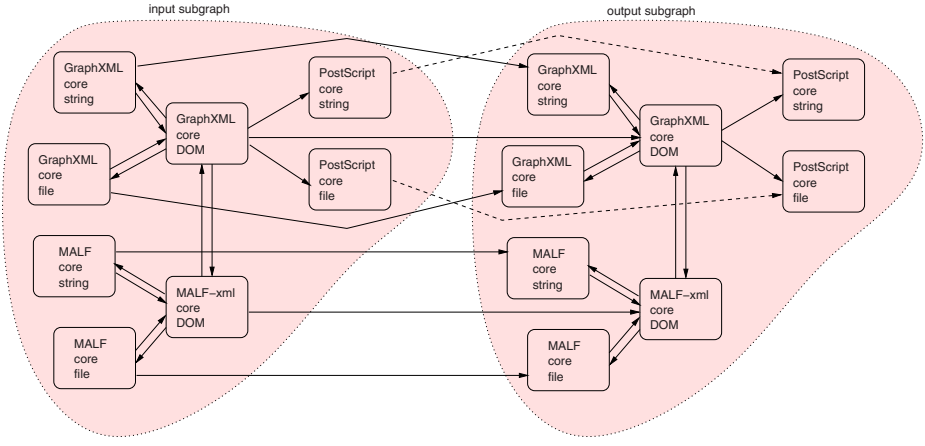

Fig. 1. An extended translation graph. Dashed lines indicate dummy merging edges.

Section 2 gives an overview of issues to be addressed by a general-purpose graph translator, section 3 presents the architecture of GraphEx, and section 4 summarizes the contributions of GraphEx and directions for future work.

2 Issues in Graph Format Translation

One of the primary issues facing a graph format translator is the fact that nearly any pair of graph formats will be incompatible in at least one aspect.

Structural mismatch refers to incompatibility of the graph structures which can be represented by the format: format A supports hierarchical graphs or multiedges, for example, while format B does not. In this case translation is impossible unless the original graph can somehow be encoded in a way that is compatible with the new format.

Information mismatch addresses incompatibility of the attributes used to specify information beyond the graph structure: format A supports attributes that format B does not, or format A places restrictions on the legal values of attributes which are different from B's requirements. A certain amount of information loss is inevitable when this occurs.

Due to the large number of formats in existence, it is not practical to require one-step translation filters for every pair of formats. A solution is to use translation sequences, so that converting from format F_0 to format F_n may involve multiple intermediate formats F_1, F_2, ..., F_{n-1}. Unnecessary information loss can result if an attribute supported by both F_0 and F_n is not supported by one of the intermediate formats F_1, ..., F_{n-1}.

Two translation sequences are involved if a drawing algorithm is used: one to translate from the user's input format F_0 to the algorithm's input format F_k, and one to translate from the algorithm's output format F_{k+1} to the user's output format F_n. In addition to within-sequence information loss, unnecessary

information loss can occur if an attribute supported by both F_0 and F_n is not supported by F_k or F_{k+1}. Furthermore, information can be lost if there is a pair of formats F_i and F_j such that F_i is converted to F_j in the first translation sequence, F_j is converted to F_i in the second translation sequence, and the translations are asymmetric i.e. if a particular attribute x of F_i is converted to attribute y of F_j in the translation $F_i \rightarrow F_j$, but a different attribute z of F_j is converted to attribute x of F_i in the translation $F_j \rightarrow F_i$.

Another issue results from extensible formats such as GML and GraphML: since users can define their own attributes, a translator cannot be guaranteed to understand all of the attributes present in a graph description. However, users often want the output from a drawing algorithm in the same format as their original input, and it is desirable for the extra application-specific attributes to be preserved. The matter is complicated by *fragile attributes* whose values may be altered the algorithm, as these attributes should *not* be preserved. Most formats do not provide an automatic way to identify fragile attributes; GML, which solves the problem by stipulating a naming convention, is one exception.

3 System Architecture

3.1 The Translation Graph

GraphEx extends the Graph Drawing Server's idea of a *translation graph*, a directed graph which describes all of the format conversions supported by the service. Figure 1 shows an example of an extended translation graph.

The translation graph is partitioned into two subgraphs, the "input subgraph" and the "output subgraph".

The input and output subgraphs each contain a vertex for each format. A format is described by a name, a version, and a form. The version primarily applies to extensible formats, and distinguishes between different attribute sets. The form distinguishes between different ways a graph in a given format may be represented: "a graph in GML format" may refer to a filename, URL, string, parsed object, etc., each of which may require different handling.

The formats supported by GraphEx can be classified into two groups: *graph-oriented* formats such as GML and GraphML, and *diagram-oriented* formats such as PostScript and GIF. Graph-oriented formats contain an explicit representation of the graph structure, while diagram-oriented formats contain only a graphical representation of a drawing; the graph structure cannot be easily extracted from a diagram-oriented format. Diagram-oriented formats can generally only be translated to other diagram-oriented formats.

Edges in the translation graph correspond to translation or merging functions, and are directed from the input format to the output format.

"Translation" refers to anything which changes the form or format of the input data — a parser is considered to be a translation because it changes the form from a filename to a parsed object, and a function downloading a remote URL is a translation because it converts the URL into a local file. Translation

edges only connect vertices within a subgraph and are duplicated in the input and output subgraphs.

Merging operations are used in the second sequence of a two-sequence translation to restore attributes of the sequence one input graph which are lost as a result of formats required by the drawing algorithm. The output of the merging step contains all of the non-fragile attributes from the original input graph plus attributes present in the current graph from the second translation sequence. Currently all unknown attributes are copied; an extension will allow the user to specify which attributes should not be copied.

Merging requires establishing a correspondence between the vertices and edges of two separate but structurally identical graphs. This is made more difficult because not all graph formats support unique vertex identifiers, even fewer support unique edge identifiers, and some formats with unique identifiers do not guarantee that the identifiers remain associated with the same object if the graph is processed in some way. Merging establishes a correspondence by attempting to match vertices according to their unique identifiers, if available. If this is not successful, the graph structure is used to determine a partial correspondence which is then refined further, if necessary, by attempting to match attributes present in both graphs. Once the vertices have been matched, a similar procedure is applied to match edges.

Merging edges connect vertices in the input subgraph with vertices in the output subgraph and are directed from the input subgraph to the output subgraph. Every pair of same-format vertices in the input and output subgraphs must be connected by a merging edge; dummy edges are included in cases where merging is not supported (e.g. diagram-oriented formats) or not implemented. To reduce the amount of work required to support merging, it is generally expected that merging will only occur between graphs in the same format; however, additional merges could be incorporated into the translation graph without modifications.

3.2 Performing Translations and Handling Mismatch

A translation sequence is found by finding the shortest path from the input format in the input subgraph to the output format in the output subgraph. Since the only connections between these two components are merging edges, the shortest path is guaranteed to contain exactly one merging edge.

GraphEx supports two options for dealing with formats which are structurally incompatible. One option is to simply not define translations between such formats, resulting in a disconnected translation graph. If the formats are truly incompatible, a "translation not possible" error is a reasonable outcome.

A second possibility is to recognize that translating from format A to incompatible format B is only impossible if those features of format A which are incompatible with format B are used. For example, if format A supports multiedges but format B does not, the translation only needs to fail if the particular graph to be converted actually contains multiedges. GraphEx allows these "partial translators" as long as the translation function generates a warning or error if the graph cannot be translated.

(The possibility of encoding the original graph G as some other graph G' which can be represented in the target format is ignored, as such an encoding is typically application-specific and thus is not appropriate for a general-purpose translator. Encoding functions can be added as algorithms to the Graph Drawing Server, if desired, and can be explicitly accessed by the user in that way.)

Unnecessary information loss due to information mismatch can be avoided by providing translations between every pair of formats, or by defining a single "most powerful format" $\mathcal{F}$ and ensuring that $\mathcal{F}$ is the only intermediate format in every translation sequence. Unfortunately, both of these solutions are impractical: both require a significant amount of work to realize, and there is always a possibility that a new or user-defined format will contain a feature not supported by $\mathcal{F}$. Furthermore, the second scheme limits the degree to which existing translation filters can be incorporated into the service because a specific set of translations is required.

Instead, GraphEx tries to minimize unnecessary information loss through edge weights and clever structuring of the translation graph.

Translation edges are assigned weights according to the quality of the translation: completely compatible formats where no attributes are added or lost have the lowest weight, followed by translations which add attributes and finally those which involve a loss of information. This ensures that the best translation sequence is chosen if there is more than one possible translation sequence for a given pair of formats.

It is also envisioned that the translation graph will evolve to contain a small core of expressive formats (such as those intended as exchange formats), with each application-specific format linked to one of the core formats. The intention is that translating from an application-specific format to one of the core formats would not involve any information loss, and that translations between core formats would involve little or no information loss.

Dummy merging edges always have weight 0 since it is not possible to merge these formats. The weights of other merging edges depend on the translation scenario.

In a single-sequence translation or in the first sequence of a two-sequence translation, merging is not necessary and thus all merging edges are assigned a weight of 0 so they do not influence the rest of the translation process.

For the second sequence of a two-sequence translation, it is also necessary to consider the translation required to convert the input for the first sequence to the input format for the merging step. To ensure that merging happens at the best possible time, merging edges are assigned a weight corresponding to the cost of this conversion. If such a translation is not possible, the merging edge is assigned a weight of ∞.

3.3 User-Defined Formats

The purpose of GraphEx is to allow a user to access a variety of graph drawing algorithms without having to worry about format translations, but a user cannot take advantage of the system if her data is not already in one of the formats

supported by the service. A new feature of GraphEx is its ability to allow users to install new formats, translations, and merges, which are then added to the translation graph and can be automatically and transparently in future drawing requests. For security, the new items are only available to the user who installed them.

Currently the user must provide implementations of the translations required to link the new format to an existing format, including any necessary parsers, but future plans include simplifying and automating the process to reduce the burden on the user.

3.4 Implementation Details

The core GraphEx service is implemented in Java; it provides an interface accessible via Java RMI. Java was chosen because of its portability — a GraphEx server can be placed on a wide variety of hosts so that it is not necessary to port existing translation filters to different environments. Java RMI also affords several advantages: it automatically handles the marshalling and unmarshalling of entire objects sent between hosts, and it supports dynamic code loading allowing Java classes to be distributed at runtime as needed throughout the system. Furthermore, RMI can easily be layered over SSL to provide secure communications between client and server.

GraphEx also supports user authentication and allows individual translations to be restricted to certain users.

There is a great deal of flexiblity as to how individual translation filters are implemented: they can be Java classes, standalone executables, shell scripts, C or C++ libraries, XSLT transforms (for XML-based formats), or anything else that can be called from a Java program. Every translation is associated with a wrapper class which hides details of the invocation of the translation filter and provides a common interface to the translation service for all translations.

A web-based frontend for GraphEx is under development. This allows the client to send a request to a webserver which then uses servlets to contact the translation service to process the request.

4 Conclusions and Future Work

In summary, GraphEx provides an Internet-accessible graph format translation service. It improves on the original translation service of the Graph Drawing Server [2] by reducing unnecessary loss of information and adding support for user-defined formats and additional security. Future work includes development of a web-based frontend and a more powerful interface to facilitate the specification of user-defined formats.

References

1. U. Brandes, M. Eiglsperger, I. Herman, M. Himsolt, and M. S. Marshall. GraphML progress report: Structural layout proposal. In *Proc. 9th Intl. Symp. Graph Drawing (GD '01)*, volume 2265 of *Lecture Notes Comput. Sci.*, pages 501–512. Springer-Verlag, 2002.
2. S. Bridgeman, A. Garg, and R. Tamassia. A graph drawing and translation service on the WWW. *Internat. J. Comput. Geom. Appl.*, 9(4/5):419–446, 1999.
3. Ivan Herman and M. Scott Marshall. GraphXML — an XML-based graph description format. In Joe Marks, editor, *Proc. 8th Intl. Symp. Graph Drawing (GD 2000)*, volume 1984 of *Lecture Notes Comput. Sci.*, pages 52–62. Springer-Verlag, 2000.
4. M. Himsolt. GML: Graph modelling language. Manuscript, Universität Passau, Innstraße 33, 94030 Passau, Germany, 1996. `http://infosun.fmi.uni-passau.de/Graphlet/GML/`.
5. A. Winter. Exchanging graphs with GXL. In Petra Mutzel, Michael Jünger, and Sebastian Leipert, editors, *Proc. 9th Intl. Symp. Graph Drawing (GD '01)*, volume 2265 of *Lecture Notes Comput. Sci.*, pages 485–500. Springer-Verlag, 2002.