Change Request #001
for the CSC444 Graph Editor

Document # CSC444-2001-CR-01A

Revision A

14th November 2001

Table of Contents

1 Scope

This change request refers to the Graph Editor Software, as specified in CSC444-2001-SRS-01B (Revision B). The change request identifies three additional new features gathered from customer feedback from initial user acceptance testing.

1.1 Reference Documents

This change request is a modification to the following document:

CSC444-2001-SRS-01B (Revision B) – Specification for the Graph Editor Software

The following documents describe the course in which this software is to be developed:

CSC444-HND-001	Course Orientation Handout
CSC444-HND-002	Notes on the Software Trading Game

http://www.cs.toronto.edu/~sme/CSC444F Course website

http://www.cs.toronto.edu/~simsuz/teaching/csc444/GXLusersguide0.1.pdf Draft GXL users guide

2 Specific Changes

A number of teams of software design experts have completed versions of the Graph Editor Software, as part of the coursework requirements for the course "CSC444F Software Engineering I". Potential end users for this software include a number of researchers in software engineering, for whom the ability to view and edit graphs is essential in their work. A cross section of the potential customer base has now been shown partial documentation and demos of the working systems, and has expressed general approval.

Indeed, the customers got very excited at the prospect of using the graph editor to assist them in their research, and immediately suggested a large number of modifications that would increase their satisfaction. The Software Mediation Executive has reviewed and filtered these suggestions, and in order to select a small number to include in the next release of the software. These are described below.

The teams participating in the course are asked to consider each of these three new features carefully, and select ***one feature*** to implement for phase 3 of the course. Selection should be based on an assessment of the cost to add each feature, and the impact the change will have on the overall size and quality of the program. Teams should be careful to document any limitations or assumptions made while implementing these alterations.

2.1 Hierarchical Graphs

The GXL standard allows for nodes in a graph to contain other nodes. This is very simply achieved by placing an entire graph between the begin and end tags of a graph element. For example:

```
<node id = "t">
      <graph id = "innerGraph" edgeid = "true" >
            <node id = "v"/>
            <node id = "w"/>
            <edge id = "e" from = "v" to = "w"/>
      </graph>
</node>
```

For simplicity, we will assume that only nodes are allowed to contain embedded graphs (although the GXL standard allows them to be embedded into edges as well). We will refer to a node that contains a graph as a *container node*, and the graph that it contains as an *embedded graph*. In software engineering, a number of common notations are based on hierarchical graphs. In some cases, the embedded graph may be shown as a separate diagram (e.g. for hierarchical dataflow diagrams). For other notations, it is usual to show the embedded graph in place (e.g. for statecharts).

To support hierarchical graphs in the graph editor software, a number of new functions will be needed:

- Read in hierarchical graphs from GXL files
- Store hierarchical graphs in the GDO
- Support two modes when viewing graphs: display embedded graphs in place, within their container nodes ("mode P"), or display embedded graphs separately, in a separate window pane ("mode S").

- Allow the user to switch between modes S and P.
- In mode P, display a graph so that all embedded graphs are shown within their container nodes.
- In mode P, selectively show or hide the embedded graphs in selected container nodes.
- In mode S, allow the user to navigate up and down the hierarchy.
- Create a new embedded graph within an existing graph node.
- Edit an existing embedded graph.
- In mode P, allow the user to select how to handle embedded graphs when generating an automatic layout:
  - compute a new layout for an embedded graph without altering the layout of the parent graph.
  - compute a new layout for the parent graph without affecting the layout of any embedded graphs.
  - compute a new layout for the graph so that all visible embedded graphs can also be re-arranged according to how they are connected to nodes outside their container nodes.
- Save hierarchical graphs in a GXL file.

For more information about hierarchical graphs, see chapter 8 of the GXL user's guide at:

http://www.cs.toronto.edu/~simsuz/teaching/csc444/GXLusersguide0.1.pdf

For an example of the use of statecharts, see lecture 13 at:

http://www.cs.toronto.edu/~sme/CSC444F/slides/L13-DesignRepresentations.pdf

2.2 Typed Nodes and Edges

The GXL standard allows nodes and edges to be typed, so that they take their appearance from the type definition. This improves consistency of appearance, and makes it easier to define notations that include several different types of node and/or edge, distinguished by their appearance. It also allows the appearance of all nodes or edges of the same type to be changed once by editing the type definition. In the GXL standard, type information is provided via the use of xlinks to a graph schema (also represented as a GXL graph). For example:

Dataflow diagram

```
<gxl>
<graph id = "Booktravel.DFD">
<type xlink:href="DFDschema.gxl">

<node id = "1">
  <type xlink:href="DFDschema.gxl#process">
  <attr name ="label">
    <string>determine form of travel</string>
  </attr>
</node>

<node id = "2">
  <type xlink:href="DFDschema.gxl#process">
  <attr name ="label">
    <string>check schedule</string>
  </attr>
</node>

<edge id = "flow1" from = "1" to = "2">
  <type xlink:href="DFDschema.gxl#dataflow">
  <attr name ="label">
    <string>travel request</string>
  </attr>
</edge>

...

</graph>
</gxl>
```

Dataflow Schema

```
<gxl>
<graph id = "DFDschema">

<node id = "process">
  <type xlink:href="gxl-1.0.gxl#NodeClass">
  <attr name ="shape">
    <string>circle</string>  </attr>
  <attr name ="borderstyle">
    <string>solid</string>  </attr>
  <attr name ="bordercolour">
    <string>black</string>  </attr>
  <attr name ="labelfont">
    <string>arial</string>  </attr>
</node>

<node id = "dataflow">
  <type xlink:href="gxl-1.0.gxl#EdgeClass">
  <attr name ="shape">
    <string>arc</string>  </attr>
  <attr name ="weight">
    <int>1</int>  </attr>
  <attr name ="linestyle">
    <string>solid</string>  </attr>
  <attr name ="linecolour">
    <string>black</string>  </attr>
  <attr name ="labelfont">
    <string>arial</string>  </attr>
  <attr name ="isDirected"></attr>
</node>
...
</graph>
</gxl>
```

The graph schema defines not only the appearance of the typed nodes and edges, but also constrains how they can be used. For example, the schema above for dataflow diagrams should constrain dataflows so that they have to connect from a process to a process, or from a process to a datastore, but cannot connect from a datastore to a datastore. Such constraints would be given by defining the appropriate edges in the schema (not shown in the example above!). For this release of the software, we will ignore this use of schemas to constrain the structure of a graph, and just concentrate on their use to define the appearance of typed nodes and edges.

We will also assume that the appearance defined in a schema can be over-ridden by providing appearance attributes for particular typed nodes or edges. For example, in the above schema for dataflow diagrams, the default colour for processes is black. However, we may wish to highlight one of the processes in our dataflow diagram by over-riding this default colour.

To support typed nodes and edges in the graph editor software, a number of new functions will be needed:

- When reading in GXL files that contain typed nodes and edges, locate the appropriate schema file, and read this in as a schema graph.
- Display a graph using information in the schema to control the appearance of typed nodes and edges.
- Change the type of a selected node (or edge) by selecting a type from those available in the schema.
- Create a new schema for a graph.
- Apply an existing schema to a graph by linking that graph to the existing schema file.
- Add a new type to an existing schema.
- Delete a type from an existing schema.
- Change the appearance of all nodes that take a particular type by editing the information in the schema.
- Change the appearance of all nodes that take a particular type by editing the attributes of one of those nodes and requesting that the change apply to the schema.
- Change the appearance of a node to over-ride the schema defaults, by editing the attributes of the individual node.
- When saving a graph that has a corresponding schema, if the schema has been edited, prompt the user to ask whether she wants to save the changes to the schema too.
- Save the changes to a schema without (necessary) saving the changes to the current graph.

For more information about types and schemas in GXL graphs, see chapter 5 of the GXL user's guide at:

http://www.cs.toronto.edu/~simsuz/teaching/csc444/GXLusersguide0.1.pdf

2.3 Collaborative Graph Editing

A key feature of software engineering is that software is usually developed by (large) teams of people. To support the use of a graph editor by teams of software engineers, the graph editing tool needs to allow different people to access a set of graphs, to see what changes other people have made to particular graphs, and to annotate graphs with comments. It should also prevent several people from simultaneously editing the same graph.

Access to a set of shared graphs can be achieved through a client-server model, where the graphs are stored on a central server, and the members of the team access them from a client workstation, running a local copy of the graph editing software. To support geographically distributed teams, the client-server architecture should support access using standard internet protocols, with password protection to prevent unauthorized access to the graphs. A standard version control system can be used to allow users to check out graphs for editing, check them back in when the edits are complete, and to store a historical trace of previous versions of each graph.

In addition to basic access to a shared set of graphs, teams need to coordinate their editing efforts. For example, it is useful to define an owner for each graph (normally the person that first creates it), and to record who created and/or edited the various elements in the graph. It is also useful to allow users to attach informal notes to elements in the graph, to record rationale information, or to capture assumptions or queries about the graph. These can all be achieved by adding attributes to a GXL graph. We can add notes to any element of a graph by using a 'struct' container attribute, with fields for author, contents, etc. We can also add author attributes to any other graph elements, to show that they were added by someone other than the graph owner.

In the following example, the graph was originally created by Frodo Baggins. Process 1 has a note added by Bilbo Baggins, while process 2 was added by Gandalf:

```
<gxl>

<graph id = "Booktravel.DFD">
<attr name = "owner">  <string>Frodo Baggins</string> </attr>
<attr name = "version">  <string>0.3b</string> </attr>
<attr name = "created on">  <string>12th November 2001</string> </attr>
```

```
<attr name = "last edited">  <string>14th November 2001</string> </attr>

<node id = "1">
  <type xlink:href="DFDschema.gxl#process">
  <attr name ="label">
    <string>determine form of travel</string>
  </attr>
  <attr name = "note">
    <attr name = "author">
      <string>Bilbo Baggins</string>
    </attr>
    <attr name = "contents">
      <string>I'm not yet sure if we want to automate this process or not!
      </string>
    </attr>
  </attr>
</node>

<node id = "2">
  <type xlink:href="DFDschema.gxl#process">
  <attr name ="label">
    <string>check schedule</string>
  </attr>
  <attr name = "author">
    <string>Gandalf</string>
  </attr>
</node>

...

</graph>
</gxl>
```

To support collaborate graph editing in the graph editor software, a number of new functions will be needed. The commands to access graphs on the server, via the version control mechanism, should be built directly into the GUI for the editor, to make access as seamless as possible. Then the following new functions will be needed:

- Provide a login and password interface for connecting to and accessing graphs on the server
- Open a graph on the server for reading only.
- Check out a graph from the server for editing.
- Browse the version history for a graph, and view previous versions.
- Record the ownership whenever a new graph is created.
- Date stamp a graph each time it is edited.
- If the current user is not the owner of the graph being edited, record the current user as author of each element of the graph added or edited.
- Allow the user to add notes attached to any node or edge of the graph.
- Indicate visually any nodes or edges that have notes attached.
- Display and hide the contents of a note on request.
- Edit the contents of a note.
- Check in an edited graph back to the server.
- Check in a new graph to the server.