Audio Engineering Society

Convention Paper

Presented at the 124th Convention
2008 May 17–20 Amsterdam, The Netherlands

An Anatomy of Graph-Based User Interfaces for Media Processing

Christopher Schultz[1], Jörn Loviscach[2], Shailendra Mathur[3], Jay LeBoeuf[4]

[1] *Work done while at Universität Bremen, 28356 Bremen, Germany; now at mediaclipping, 28217 Bremen, Germany*

[2] *Hochschule Bremen (University of Applied Sciences), 28199 Bremen, Germany*

[3] *Softimage Corp., Avid Technology, Inc., Montreal, Quebec H2X 2V2, Canada*

[4] *Work done while at Digidesign, Daly City, CA 94014, USA; now at Imagine Research, Inc., San Francisco, CA 94127, USA*

Correspondence should be addressed to Christopher Schultz (`cschultz@tzi.de`)

ABSTRACT

Graph-based user interfaces are employed in a variety of software such as audio synthesizers, video compositing tools, and database application builders. All of these uses afford the graphical metaphor of a graph: "Nodes" such as sound generators or filters are tied together by "links," which may represent signal flow or conceptual relations. Focusing on media production tools, we have examined a large range of current software products to find out which de-facto standards have evolved in the field of graph-based interfaces and which features can be considered unique. We categorize a multitude of interface concepts employed in actual graph-based interfaces and describe differences in their implementation. The findings provide guidelines for developers of media production software.

1. INTRODUCTION

In a variety of domains, graphs are used as metaphors to describe structures or relations. Examples include subway maps, molecular structures, flowcharts of production processes, and organizational charts of management hierarchies. In multimedia production software, graph-based user interfaces are commonly used to represent and control the processing of audio or video data. In most of these cases, graphs can be considered a UI metaphor [1]. Only in diagramming tools, graph-based interfaces are used to create graphs as such; elsewhere, graphs

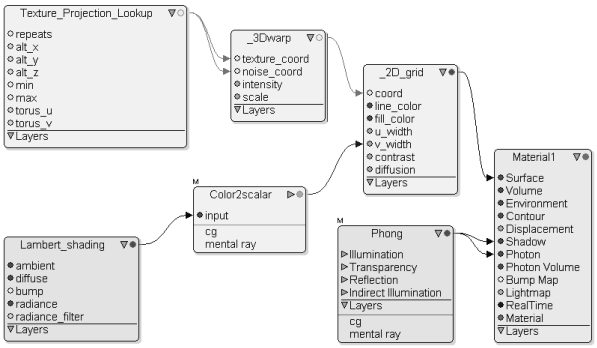

Fig. 1: In Softimage XSI, the ports on a shader's nodes are color-coded according to which data type they handle.

are analogies representing something different, such as a flow of data. This is similar to the desktop trash can, which is not an actual trash can, but yet another UI metaphor.

Mathematically speaking, a graph [2] is a structure that models relationships within a set of entities called "nodes" by connecting these nodes via "edges." In a diagram, the nodes are usually represented as boxes or icons; the edges are drawn as straight or curved connection lines. In a *directed* graph, the relationship between the nodes is not symmetric, but of a send/receive-type. This is visualized by drawing each edge as a line with an arrow. Directed *acyclic* graphs form an important subtype: Walking in the arrows' direction, one can never visit the same node twice.

Note that most of the applications do not conform strictly to the common mathematical definition of a graph, since they distinguish between different types of sockets per node to which edges can connect. For instance, in 3D software, one socket of a node may control the overall color of a material; another socket of the same node may control the size of the highlights (see Figure 1). This issue is solved by distinguishing between the nodes the sockets they contain (called "vertices" or "ports"). Which connections are allowed may be described with the help of a graph grammar [3].

To study graph-based interfaces, we borrowed an approach from other scientific disciplines such as zoology, which seeks to identify, categorize, and chart existing species. Rather than build and evaluate an application from scratch, we attempted to learn about the interfaces with which users are already familiar: Which functionality has evolved as a de-facto standard? What are the variations to be seen? Since conforming to what the user expects is a vital issue in interface design, the answers to such questions contain important information needed to create new or to improve existing software.

When creating applications that merely consist of menus, text windows and dialog boxes, developers are relieved from considerations of this kind: Such issues are exhaustively handled through standardized frameworks for graphical user interfaces such as Sun Java Swing. There is not yet an established toolkit, however, for graph-based user interfaces. Thus, the developer is left alone with the task of identifying interaction modes and implementing their details. As our analysis reveals, in this process it is easy to overlook seemingly obvious features such as panning.

2. TYPOLOGY

Researchers have proposed graph-based interfaces for an extreme variety of applications. In his seminal work of 1988, Haeberli [4] introduced ConMan, a graph-based UI that allows combining function modules to form interactive graphics software. Digidesign TurboSynth, dating from the same year, employed graphs for audio synthesis. Softimage Eddie, published one year later, was the first commercial software to apply graphs to effects and compositing processing for digital film and video production. Such software paved the path for most of today's graph-based UIs. The manufacturers have polished these interfaces continuously, so there is much to learn from the existing products.

We were able to classify each application of graph-based interfaces found in the marketplace into one of three basic types, contrary to the richness of applications found in scientific literature. This may indicate a pressure toward standardization. The three types we found are the following:

- **Graph exploration.** This software employs user interfaces to visualize and explore graphs. Possible fields of application are the analysis of relations of database structures, Internet content, or biological molecular structures. These

tools automatically layout graphs in 2D or 3D space and allow the user to explore the graph interactively by panning, zooming, and possibly also rotating. However, they do not offer functions to edit the graph itself.

- **Graph modeling.** These applications offer special interfaces to let the user create and edit graphs visually. Software of this type is often applied to document or present abstract relations or processes as drawings such as flow charts, mind manager trees, UML class and activity diagrams, or organizational charts. In contrast to user interfaces of the graph exploration type, graph modeling tools offer interface features for interactive editing; they support manual changes in the design style of nodes and edges.

- **Process control.** This software employs a graph-based user interface to control an underlying process such as the flow of audio and control signals within a modular software synthesizer. Commonly, the UI enables the user to perform operations that manipulate the graph directly, like placing a new node on the canvas, moving or deleting it, connecting to or disconnecting it from other nodes. All changes have a direct influence on the underlying process, so that the user interface can be understood as a front-end control for a process running in the background. Hence, this kind of user interface can be considered a visual programming language [5]. This type of application is found most often in media production software; hence, we focus on it in this paper.

3. INTERFACE FEATURES

To study the features of graph-based interfaces for process control in detail, we collected a broad cross-section of popular software. The list of 14 analyzed applications includes software for audio synthesis and processing such as Cyclin '74 Max/MSP and Propellerheads Reason 5 as well as video compositing, 3D animation, and mathematical simulation.

The study covered 27 criteria which were structured into five groups: node features, connection features, view options, graphical aspects, and layout options. In every category we also looked for special, uncommon features.

3.1. Node Features

Nearly all analyzed UIs (13 of 14) distinguish visually between different types of nodes. Half of the UIs (7 of 14) indicate the node type through a corresponding background color of the node. Diverse node shapes can be seen in four applications. In another four, node parameters are directly accessible inside the graphical node object.

A significant majority of UIs (10 of 14) enable the user to edit the name of a node element directly within the graph view. Two major techniques are used: The user can click with the left mouse button on the node's name label or the UI requires a click with the right mouse button on the node object to open a context menu. In both cases, the node's name label turns into a text input box.

All analyzed UIs highlight selected nodes through a change in color or through a colored frame when they are selected by a single mouse click or by drawing a selection using the mouse. Many UIs (9 of 14) reveal a node's properties such as filter parameter settings when the node is double-clicked.

Many UIs offer one of three major methods to place a new node object on the screen. Numerous UIs even support several of these:

- Clicking with the right mouse button on an empty place of the canvas opens a context menu from which the user can select the command to create a new node. (8 of 14)
- The toolbar offers a specific icon to click on. (7 of 14)
- The standard menu offers a specific entry. (9 of 14)

A significant majority (12 of 14) of the tested UIs allow the user to copy and paste node objects using shortcut keys.

A majority of UIs (9 of 14) do not allow nodes to be scaled in size. Five UIs allow scaling by dragging a corner handle of a node. In all of them, only the shape of the node is scaled, whereas the node's content (e.g. labels or icons) retains its size.

Eight of the 14 tested UIs offer to group node elements into a nested node object, which may reduce visual clutter. This may be invoked via a shortcut key (7), a context menu (6), or a menu command (3). In a remarkable coincidence, all five examined video compositing tool UIs include the grouping feature, accessible via a shortcut key and a context menu.

Several UIs (4 of 8) offer a special type of node nesting: The user may encapsulate nodes into a module that can even be stored and reused. Thus, users can create their own meta-nodes.

3.2. Connection Features

The node elements of nearly all tested UIs (12 of 14) include sockets as small graphical marks acting as input or output interfaces for connection lines. Half of those 12 UIs highlight a socket when the mouse hovers over it. When the mouse hovers above a connection line, the line is highlighted by 6 of the 14 tested UIs.

The most common way to connect two nodes is to click on the source node or exactly on its output socket, hold the mouse button, drag a line to an appropriate socket of the destination node, and release the mouse on it. Twelve applications feature sockets that can create connections this way. However, when nodes offer several input and output channels (2 of 14), the user additionally has to select the wanted channel during this process. On top of that, several UIs offer special connection features:

- "Auto connect": If the user drags a node object close to another node, some UIs (2 of 14) automatically suggest a connection between their sockets (e.g., by drawing a transparent line). This connection is established when the mouse button is released.
- "Auto insert": If the user drags a free node over an existing connection line and releases the mouse, some UIs (5 of 14) will automatically insert this node into the process chain. For this to happen, some UIs require pressing a certain key while releasing the mouse.

We found three major ways to disconnect nodes. Most commonly (7 of 14), the user clicks on a socket or line arrow to lift the line and then drags it to an empty area of the canvas; the line is removed when the user releases the mouse. Alternatively (5 of 14), the user selects a line by clicking on it, and hits the Delete or the Backspace key. On a click with the right mouse button, some UIs (4 of 14) present a context menu with—among others—a delete command. Furthermore, several UIs (4 of 14) allow the user to extract nodes from the graph through key shortcuts.

As a special feature, 2 of 14 UIs offer a mouse gesture called "shaking": When a user drags a node quickly to the left and to the right, it is released from the process chain. We found no other usage of mouse gestures in the range of software examined.

Most UIs (12 of 14) detect when a new connection would lead to a loop within the process graph; they refuse to create such a connection that would cause the graph to cease being acyclic.

3.3. View Options

A large graph may require the user to pan over it, which is supported by nearly all UIs (13 of 14). We identified several techniques for performing a pan action: clicking and dragging with the mouse while pressing a certain key (8), using the scrollbars (7), or pressing the middle mouse button and dragging the mouse (4).

A significant majority of the tested UIs offer a zoom feature (11 of 14), similar to other zoomable user interfaces, which are of particular importance to mapping applications [7]. The user can zoom in and out on the displayed graph, which is especially helpful for large graphs. However, there seems to be no common UI concept for zooming in or out:

- Several UIs (5) offer shortcuts, mostly based on the Plus and Minus keys.
- A couple of UIs (5) require the user to press a special key and additionally click with the mouse. Thus, the user can indicate the center of the zoom operation.
- Some UIs (5) employ the mouse wheel for zooming.
- A few UIs (2) include a zoom command in the menu or toolbar.

In order to offer the user an overview of a large canvas, half of the UIs (7 of 14) include a bird's eye view along with the standard display, as known from image editing tools and video games.

Some applications (6 of 14) enable the user to choose the amount of information displayed in node objects. This feature allows users to reduce the visual complexity of a graph-based UI in certain situations and to show all node details in other situations. There are mostly two or three levels of detail available. In most of these cases (5 of 6), the user can toggle the interface between showing all node sockets and showing only the used sockets. Fewer UIs (2 of 6) allow the user to enable or disable node icons or thumbnails.

The majority of the analyzed UIs (9 of 14) provide a visual grid on the canvas which can help users to layout the nodes by hand. Nearly all UIs featuring a grid (8 of 9) possess a mode to automatically snap node elements to the grid.

As a special feature, some UIs (2 of 14) allow users to define customized perspectives, each of which describes a visible section of the canvas and the graph objects. Via key shortcuts, users can quickly move to stored views.

3.4. Graphical Aspects

The basic node shape used by most UIs (10 of 14) is a box, oftentimes with smoothly rounded edges (4 of 10). However, many UIs employ different shapes for different node types, so that also circles (4 of 14) or capsules (3 of 14) are available as well.

The majority of UIs draw connections between nodes as straight lines (10 of 14). Other line types are Bézier curves (5 of 14), as well as straight lines running in horizontal or vertical direction (3 of 14). Several UIs allow the user to set the preferred line type.

There is a clear standard concerning the canvas' color: 10 of 14 tested UIs use a gray background. All UIs (9 of 14) that offer a grid—to be used as a placement aid—draw this grid in a gray tone. For the connection lines, however, there is no de-facto color convention: some UI draw them black (6 of 14), others white (4 of 14). Furthermore, several UIs have multiple line colors which indicate the data type flowing through this line (4 of 14).

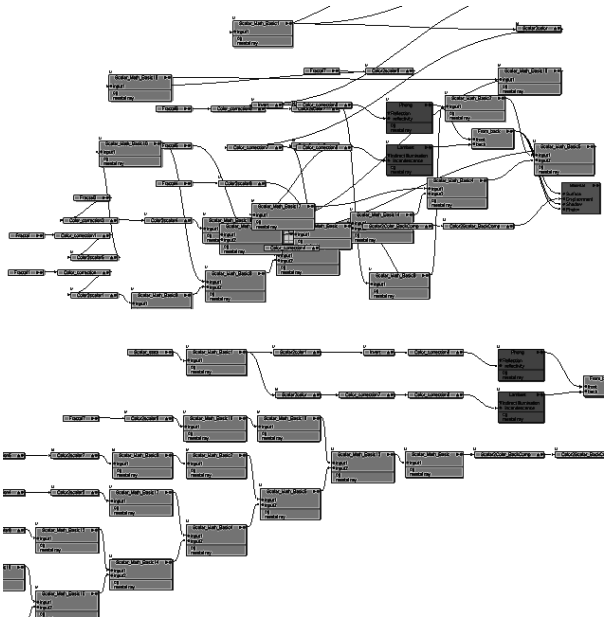

Fig. 2: Automatic graph layout may (bottom) be employed to clean up the workspace such as here in Softimage XSI.

One of the tested UIs offers an option in order to toggle between two graphical display types. In an "enhanced view" mode, further information about node objects or connection elements is indicated graphically; for instance different line styles may represent the data type of a connection.

3.5. Layout Features

Most graph-based UIs enforce a certain flow direction, due to the location of the sockets on the node objects. If the input sockets are placed at the top of the node and the output sockets at its bottom, a less-cluttered graph will require the general flow to be from top to bottom. Such a top-down flow only is favored in 5 of the 14 tested UIs. The majority (11 of 14) shows a left-to-right flow direction by default. Some UIs even allow the user to set the preferred flow direction in the options (3 of 14).

Much research has addressed the automatic layout of graphs [6], the objective being a clear arrangement of the nodes with only a minimum number of crossings of their edges, see Figure 2. Less than half of the UIs (6 of 14) provide this feature—notably only video compositing and shader design tools.

Two of these applications additionally offer a compact layout that arranges the graph with smaller gaps between the node objects, allowing larger graphs to fit onto the screen. One of the examined UIs aids in visual grouping: A key shortcut attracts selected nodes to the rest of the graph or repels them from it.

4. CONCLUSION

Many media-driven applications employ graph-based user interfaces in order to represent and control the processing of audio or video data. Based on a testing pool of different graph-based applications, we analyzed different software interfaces. We looked into vital functions such as the automatic layout of the graphs. But we also looked into interface details that seem to be marginal, but are used so often that they are equally critical for the user experience (e.g., the specific interaction modes supported to disconnect one node from another). We classify the interface features we found and give statistical numbers on which specific solutions are prominent.

For most examined features, we found a large degree of commonality in these user interfaces, but also noted some unique and often promising approach in one or two products. Both kinds of observations can be employed as a cookbook by software developers. Many of the discrepancies found (such as the lack of a panning function) look like issues to be solved in the next version of the corresponding software. It comes at some surprise, however, that only a minority of the UIs offers an automatic layout function [6], even though this has been a topic in research for decades. A detailed analysis such as ours would have prevented such issues at the outset.

Finally, we hope that our observations form a stepping stone for future research into the usability of these features, helping to further improve the look and feel of graph-based user interfaces.

5. ACKNOWLEDGMENTS

Jrn Loviscach's work was partly funded by grant 1742B04 of the German Ministry of Education and Research (BMBF). The views and conclusions contained herein are those of the authors.

6. REFERENCES

[1] Alan F. Blackwell, "The reification of metaphor as a design tool," ACM Transactions on Computer-Human Interaction, Vol. 13, No. 4, 490–530, New York, USA, 2006.

[2] Jonathan L. Gross and Jay Yellen, "Handbook of graph theory," CRC Press, 2003.

[3] Jun Kong and Kang Zhang and Xiaoqin Zeng, "Spatial graph grammars for graphical user interfaces," ACM Transactions on Computer-Human Interaction, Vol. 13, No. 2, 268–307, New York, USA, 2006.

[4] Paul E. Haeberli, "ConMan: a visual programming language for interactive graphics," ACM SIGGRAPH Computer Graphics, Vol. 22, No. 4, 103–111, 1988.

[5] Marat Boshernitsan and Michael Downes, "Visual Programming Languages: A Survey," EECS Department, University of California, Berkeley, Technical Report No. UCB/CSD-04-1368, 2004.

[6] Ivan Herman and Guy Melancon and M. Scott Marshall, "Graph visualization and navigation in information visualization: A survey," IEEE Transactions on Visualization and Computer Graphics, Vol. 6, No. 1, 24–43, 2000.

[7] Kasper Hornbæk and Benjamin Bederson and Catherine Plaisant, "Navigation patterns and usability of zoomable user interfaces with and without an overview," ACM Transactions on Computer-Human Interaction, Vol. 9, No. 4, 362–389, 2002.