Visualizing Flow Diagrams with SHriMP

Derek Rayside and Marin Litoiu
IBM Centre for Advanced Studies
IBM Toronto Laboratory
{drayside, marin}@ca.ibm.com

Margaret-Anne Storey and Casey Best
Department of Computer Science
University of Victoria
{mstorey, cbest}@csr.uvic.ca

1 Introduction

This workshop paper describes our work on visualizing *message-flow diagrams* with SHriMP.

Message-flow diagrams are a visual formalism for describing the movement of data between information systems, and may be drawn with IBM's MQSI (Message Queue Systems Integrator) product. An associated runtime system executes these diagrams: i.e. combines, transforms, and delivers the messages. MQSI is typically used to integrate information systems when large corporations merge. Much of IBM's flow related middleware is developed in Toronto.

The SHriMP (Simple Hierarchical Multi-Perspective) visualization technique has been designed to enhance how people browse and explore complex information spaces. It was originally designed to enhance how programmers understand programs [2, 3]. SHriMP presents a nested graph view of a software architecture. Program source code and documentation are presented by embedding marked up text fragments within the nodes of a nested graph. Finer connections among these fragments are represented by a network that is navigated using a hypertext link-following metaphor. SHriMP combines this hypertext metaphor with animated panning and zooming motions over the nested graph to provide continuous orientation and contextual cues for the user.

SHriMP was originally developed for visualizing the static structure of programs, and later was applied to visualizing Protege knowledge bases. We are interested in using SHriMP's advanced visualization features, namely its smooth zooming capabilities within a nested graph and advanced layout algorithms, to visualize MQSI flow diagrams. These diagrams are basically 'box and arrow' pictures, with one key difference: the nodes have *terminals* or *ports* where the arcs are attached to them. These terminals have specific meaning and are an integral part of the picture. This paper focuses on the changes that we have made to SHriMP to accomodate these terminals.

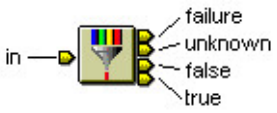
Submitted to the OOPSLA 2001 Software Visualization Workshop organized by Wim de Pauw, Steven Reiss and Gary Sevitsky.

2 Flow Visualization in MQSI

One of the important characteristics of nodes in message-flow diagrams is that they have specific *terminals* where arcs may be attached to them. Which terminal an arc is attached to is fundamental to the meaning of the diagram, as discussed below, and so these terminals must be explicitly represented in some fashion.

A Simple Example Figure 1 shows an example of a 'filter' node from the MQSI flow editor with its terminals labelled. A filter node is a kind of primitive node that is analogous to a simple conditional test in an imperative programming language.

Figure 1 A Filter Node from MQSI



Filter nodes have one input terminal and four output terminals: **failure**, **unknown**, **true** and **false**. When a message arrives at a filter node, its condition is evaluated, and a message is sent from one of the output terminals depending on the result of the evaluation.

Other kinds of nodes may accept multiple inputs and send multiple outputs.

Terminals Have Distinct Identity Terminals have distinct identity: that is, they cannot be identified with arcs as arrowheads can, nor can they be subsumed by the nodes they are attached to. Terminals are different than arrowheads in two ways: terminals are still present on a node even if there are no arcs connected to

them; and multiple arcs may be connected into a single terminal.

Terminal Arrangement MQSI has one main feature for arranging the terminals on nodes: *rotation*. Figure 2 shows a filter node in the *left-to-right* rotation and in the *bottom-to-top* rotation. The main purpose of this feature is to facilitate drawing cycles with a minimum of line crossings.

Figure 2 Node Rotation in MQSI

Currently, the user can (and must) specify the desired rotation for a node: it is not computed by any of the available layout algorithms (discussed below). When a node is rotated, the output terminals are always opposite to the input terminals.

Bend Points in Arcs The second important visual feature in MQSI is *bend points* in arcs. As with node rotation, bend points are mainly used to facilitate cycles. Discussions with Grant Taylor and Evan Mamas of the IBM MQSI Group have informed us that *bend points* in arcs are considered an essential feature by many users. SHriMP does not currently support bend points, but we are considering adding this functionality.

Layout Algorithms MQSI currently features two simple layout algorithms: *left-to-right* and *top-down*. SHriMP currently features *Sugiyama* and *Spring*. None of these algorithms are particularly well suited to the cyclic nature of flow diagrams, and so we intend to investigate other layout algorithms.

3 Terminals

There are a number of issues to consider with regard to the arrangement of terminals; we have named some of these: *posture*, *grouping*, *orientation*, and *placement*. Each is discussed below with an explanatory diagram.

Posture Posture refers to the placement of the output terminals with respect to the input terminals. There are two main alternatives: *fixed* and *flexible*, as pictured in Figure 3. Fixed means that the output terminals are always opposite the input terminals, as is currently done in MQSI. Flexible means that the terminals may be positioned on any side, as long as each side contains either only input terminals or only output terminals.

Figure 3 Posture: Fixed vs Flexible

We intend to use the flexible posture in SHriMP although SHriMP currently supports only the fixed posture (and does not currently support rotation either).

Grouping Grouping refers to whether the input terminals are all on one side and the output terminals all on another. Figure 4 shows terminals grouped on the left and not grouped on the right. MQSI currently groups terminals. SHriMP also currently groups arcs: all arcs go from the centre of the bottom of the source node to the centre of the top of the target node.

Figure 4 Grouping: Together vs Independent

We will not group terminals in SHriMP as this will allow the user or a layout algorithm to minimize superfluous line crossings in the drawings.

Orientation Orientation refers to the angle of the terminal: it may be *orthogonal* to the edge of the node, or it may be *angled* to be in line with the arc connected to it. Figure 5 shows an orthogonal terminal on the left and an angled terminal on the right.

Figure 5 Orientation: Orthogonal vs Angled

Angled terminals appear more like arrowheads, which has some intuitive appeal. However, as can be seen in Figure 5, angling tends to involve an unacceptable level of distortion when the terminals are represented as images. Moreover, angling only makes sense if there is only one arc connected to the terminal. Since it is very likely that there may be more than one arc connected to a terminal, we will use orthogonal terminals.

Placement Placement refers to the placement of the terminal along the node's edge: is it the *default* place-

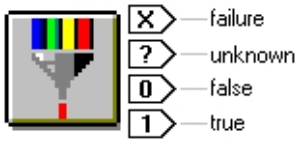
ment (probably the centre of the side), or is it the *closest* placement (i.e. the placement that would put the terminal as close as possible to the node at the other end of the arc). The default placement is depicted on the left of Figure 6, and the closest placement is depicted on the right.

Figure 6 Placement: Default vs Closest

Both SHriMP and MQSI currently support only default placement (and SHriMP is limited to the top and bottom sides). Closest placement may reduce line crossings, especially if there are multiple terminals placed along one side. One complication of closest placement algorithm is that it is based on the assumption that there is a single target to be closest to, and this may not be the case if multiple arcs are connected to the same terminal. We intend to use closest placement and to devise some heuristics to deal with this complication.

Visually Differentiated Terminals MQSI currently groups terminals in the default placement, and this consistency of placement assists the user in knowing which terminal is which: for example 'the bottom terminal is always the true-output terminal'. The more flexible terminal arrangements described above require some other way for the user to identify nodes. Mike Beltzner of IBM's UCD group has suggested simply using different icons for each kind of terminal. Figure 7 shows a filter node (as in Figure 1), with the output terminals each indicated with a different icon: '**1**' indicates true; '**0**' indicates false; '**?**' indicates unknown/undefined result; and '**X**' indicates failure in computation.

Figure 7 Visually Differentiated Terminals



Of course there are other ways in which terminals may be visually differentiated from each other besides iconography: e.g. colour, shape, etc.

4 Visualizing Flows in SHriMP

Figure 8 depicts how arcs are currently connected to nodes in SHriMP. Figure 8 shows a SHriMP node with three incoming arcs and two outgoing arcs. Arc direction is distinguished by where the arc connects to the node: incoming arcs connect to the top and outgoing arcs connect to the bottom. This convention sometimes causes arcs to be drawn over the nodes they are connected to, as is the case with one of the outgoing arcs in Figure 8. Arcs crossing over nodes in this way could be avoided if the arcs were not attached to fixed positions on the nodes.

Figure 8 SHriMP Node (current presentation)

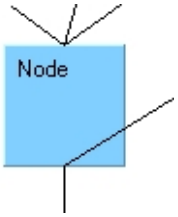
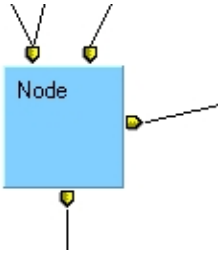


Figure 9 illustrates the desired presentation of a SHriMP node with terminals, according to the requirements discussed in the previous section. The terminals are independent from one another (not grouped), orthogonal (rather than angled), the posture is flexible (instead of fixed), and they are placed in the closest position to connected nodes (therefore minimizing the length of the arc). The two requirements that are not shown in Figure 9 are visually differentiated terminals and node icons. Notice that this more flexible positioning of terminals may prevent arcs from crossing over the nodes and the terminals that they are connected to.

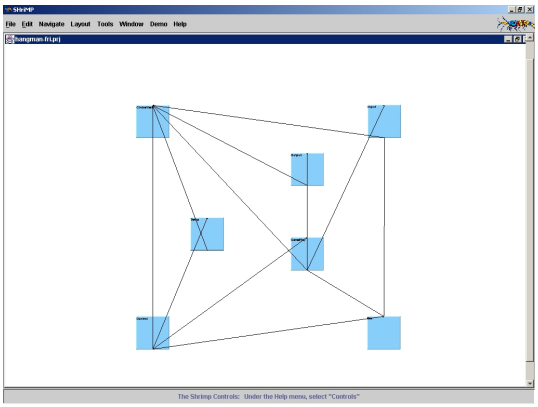
Figure 9 SHriMP Node (desired presentation)



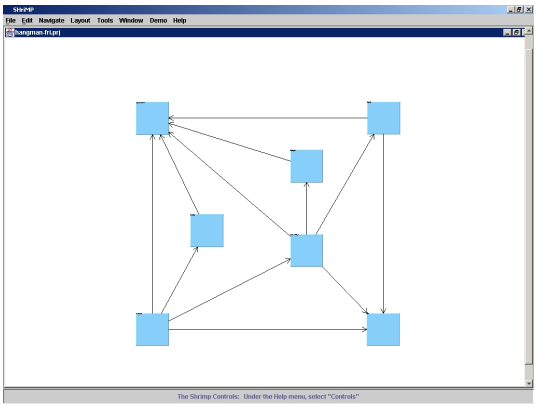
To accommodate these requirements for flow diagrams, three changes were made to arc drawing in SHriMP: flexible arc endpoint positioning on nodes, calculation of minimal arc lengths between nodes, and arrowheads. Arrowheads are not strictly necessary for visualizing flow diagrams because the terminal images also indicate direction. However, the arrowheads are necessary to indicate direction when using flexible arc endpoints and visualizing data from other domains (e.g. programs). These three changes have not degraded performance in a noticeable way.

Figure 10 shows how these three changes result in

Figure 10 SHriMP before and after changes to arcs



a) arc direction indicated by connection position (top=incoming, bottom=outgoing)



b) arc direction indicated with arrowheads; allows for flexible arc endpoint positions and shortest path arcs

clearer SHriMP views: all line crossings have been eliminated. Notice that both Figure 10a and Figure 10b have exactly the same nodes in exactly the same locations and exactly the same arcs: all that differs is where the arcs are connected to the nodes. Notice also that the graph displayed in Figure 10 is obviously planar, so no line crossings are necessary from a graph theoretic perspective. However, all arcs except one cross over the nodes they connect to when direction is indicated by where the arc is connected to the node (Figure 10a). Both SHriMP and RIGI [1] were originally designed with the convention illustrated in Figure 10a.

Our current implementation of these changes to the arc rendering in SHriMP does not deal with the case when multiple arcs are connected to a single terminal. Solving this problem will require developing some heuristic to determine where to place these terminals to minimize arcs crossing over the nodes they are connected to. One possibility is to assign weights to nodes based on proximity or interest to the user. Regardless of the heuristic chosen, in some cases arcs will cross over the nodes they are connected to (e.g. if the graph is not planar). This inevitability can be compensated for by allowing the user to manually reposition terminals according to individual preference.

5 Conclusion

Visualizing MQSI message flow diagrams in SHriMP has proven to be a very interesting challenge that has caused us to improve SHriMP. Specifically, we have added the possibility of having *terminals* on nodes, since these are required to visualize flow diagrams. Terminals are a visual feature that communicate extra information about nodes and the relations between them. We have also improved the way SHriMP connects arcs to nodes in diagrams that do not use terminals (see Figure 10).

Although the concepts of terminals have not been used extensively in program visualization, after our experience with flow diagrams we think that they hold some promise. One simple idea is to use terminals to categorize arcs: e.g. there could be a terminal for data related arcs and another for control related arcs. In flow diagrams terminals have semantics that are independent of the kind of arcs connected to them. One way to use terminals and retain this independence in program visualization is to have terminals representing normal and exceptional exit paths, as is done with filter nodes in flow diagrams (see Figure 1). We intend to explore the use of terminals for program visualization in future work.

References

[1] H.A. Müller and K. Klashinsky. Rigi — A system for programming-in-the-large. In *ICSE'88*, pages 80–86, Raffles City, Singapore, April 1988.

[2] M.-A. Storey, H.A. Müller, and K. Wong. *Manipulating and Documenting Software Structures*, pages 244–263. World Scientific Publishing Co., November 1996. Volume 7 of the Series on Software Engineering and Knowledge Engineering.

[3] J. Wu and M.-A. Storey. A multi-perspective software visualization environment. In *CASCON'00*, pages 41–50, Toronto, November 2000.