version

1.1

BILKENT UNIVERSITY

i-Vis Information Visualization Research Group

Chisio Programmer's Guide

BILKENT I-VIS RESEARCH GROUP

Chisio Programmer's Guide

Computer Engineering Department • Bilkent University
Ankara 06800, TURKEY
Phone +90 312.290.1401 • Fax +90 312.266.4047

Table of Contents

Overview

In this chapter, we give an overview of the ways in which Chisio can be customized by a programmer.

Chisio is a graph visualization tool for creating, editing and layout of *compound* or *hiearhically structured* graphs. The tool features user-friendly interactive creation and manipulation of compound graphs. In addition, a number of popular layout algorithms are provided.

Chisio 1.1 is based on the Eclipse Graph Editing Framework (GEF) version 3.1, written in Java 1.5. The tool Web page including other documentation on Chisio as well as this one follows:

http://www.cs.bilkent.edu.tr/~ivis/chisio.html

Use Cases

Chisio can be used for different purposes. If you would like to simply use Chisio as a generic graph editor, please refer to the *Chisio User's Guide*. However, if your goal is one of the following by programming on to the existing Chisio framework, then continue reading this manual:

- Customize the graph editor for a specific application (e.g. one that is used to draw UML class diagrams or a tool for visualization of social networks);
- Customize the tool for implementing a new layout algorithm (e.g. an algorithm that you developed and would like to test in an interactive tool).

Before you start the customization, you might like to read the following documents on Graphical Editing Framework (GEF).

- Main documentaion page for GEF:

  http://www.eclipse.org/gef/reference/articles.html

- An overview about the MVC architecture widely used in GEF:

  http://www-128.ibm.com/developerworks/opensource/library/os-gef/

- An example for displaying a UML Diagram:

  http://www.eclipse.org/articles/Article-GEF-Draw2d/GEF-Draw2d.html

GEF and Chisio Model

GEF assumes that you have a model that you would like to display and edit graphically. The controllers bridge the view and model (Figure 1). Each controller is responsible both for mapping the model to its view, and for making changes to the model. The controller also observes the model, and updates the view to reflect changes in the model's state. Controllers are the objects with which the user interacts.

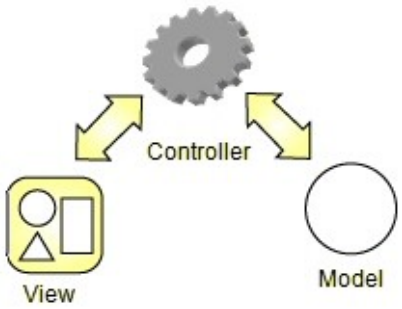

Figure 1 Model-view-controller structure used in GEF [http://www.ibm.com/developerworks/opensource/library/os-gef]

The compound graphs are modeled in Chisio using this framework with the classes shown in Figure 2. A compound node manages a list of children graph objects. A dedicated one is set as the root of the nesting hierarchy. Through recurvise use of compound nodes as child objects, an arbitrary level of nesting can be created.

Nodes, edges and compound nodes in Chisio all have distinct properties and UIs, which can be changed by its graphical user interface or through programming. Each node is drawn as a rectangle, ellipse or triangle. Edges can be drawn in a variety of styles. Compound nodes are always drawn with a rectangle, where the name text is displayed on its bottom margin and its geometry is auto-calculated using the geometry of its contents to tightly bound its contents plus user-defined child graph margins. Figure 3 shows the basics of drawing compound graphs in Chisio.

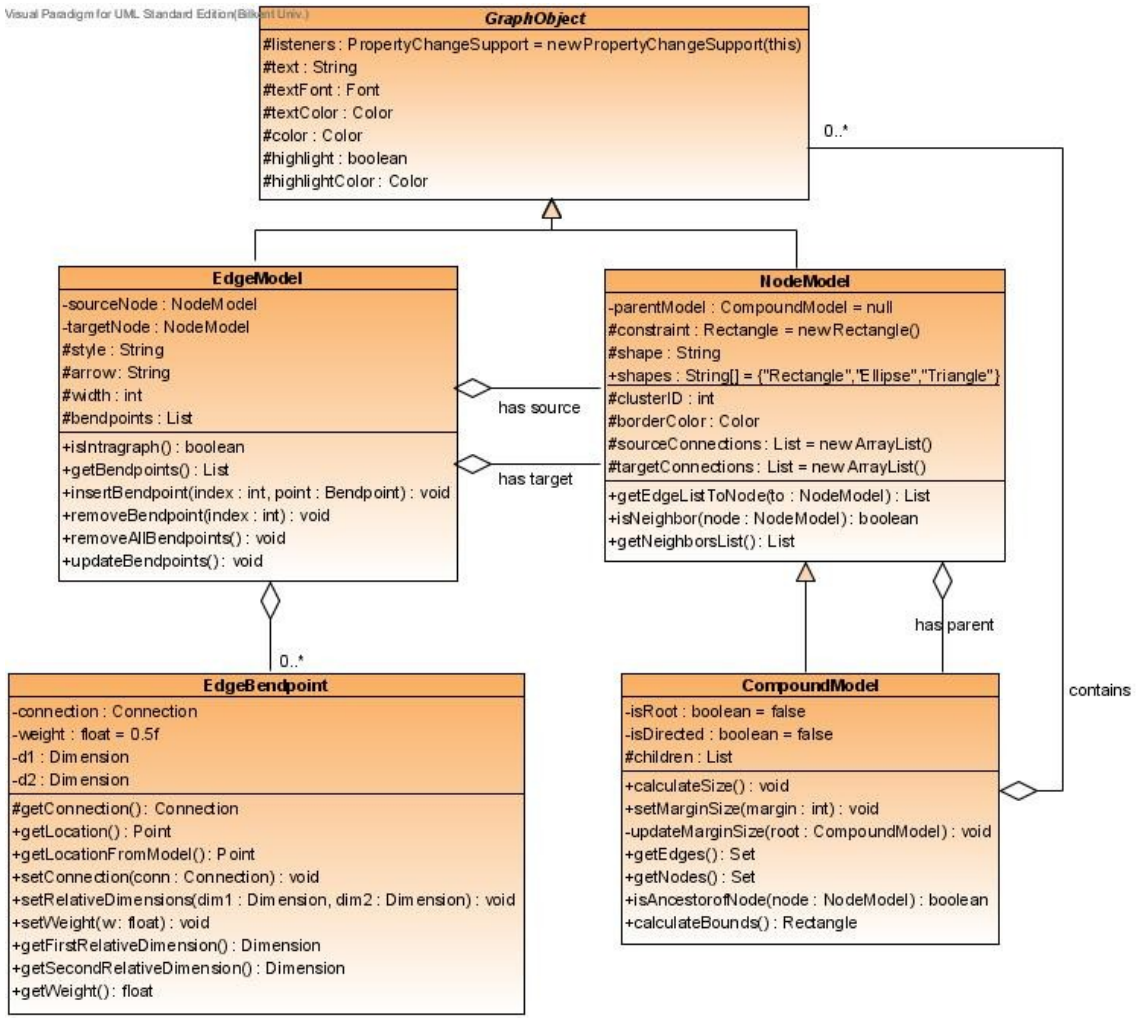

Figure 2 A UML Class Diagram illustrating the compound graph model used in Chisio

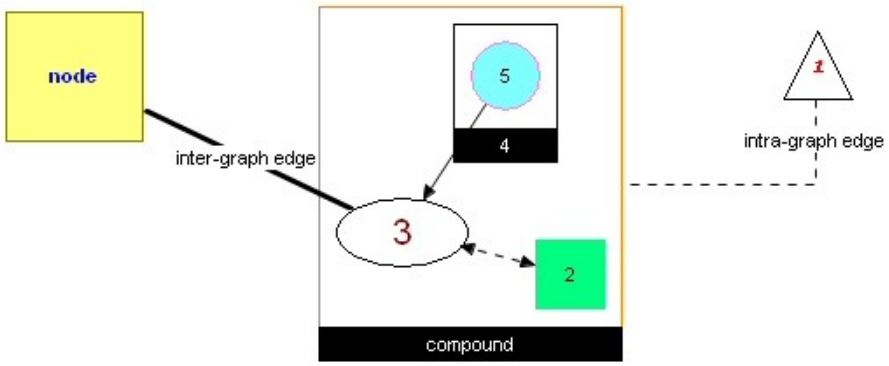

Figure 3 Basics of drawing in Chisio

Chapter 2

Customizing Chisio for a Specific Application

In this chapter, we describe how one can customize Chisio for their specific application through programming on to the existing Chisio framework.

One can customize Chisio for their specific needs by adding new node/edge types or by modifying existing nodes/edges (UI and attributes). In addition, you may customize menus to add new functionality as well as modifying node and edge menus and property inspectors.

Customizing User Interface

Adding new shapes

There are several shapes available for nodes in Chisio: *Rectangle*, *Circle* and *Triangle*. These shapes may be sufficient for some applications but many applications will require different types of shapes for drawing graphs of your own. In this part, we provide an example on how you can add a new shape to Chisio nodes.

If you want to put an image as a figure, then you should create a figure which draws an image file. For example suppose there is an image file "node.bmp" in C: drive. We use it as a node shape in the following figure class.

```
import org.eclipse.draw2d.*;
import org.eclipse.draw2d.geometry.*;
import org.eclipse.swt.graphics.Image;

public class ImageFigure extends Figure
{
    public ImageFigure()
    {
        super();
    }

    public ImageFigure(Rectangle rect)
    {

        setBounds(rect);
    }

    protected void paintFigure(Graphics g)
    {
        Rectangle r = getParent().getBounds().getCopy();
```

```
        setBounds(r);
        Image image = new Image(null, "C:\node.bmp");
        g.drawImage(image, r.getLocation());
    }
}
```

But if you want to draw your own shape, then create your own figure by providing the code for painting it. Our example shape is a *diamond*.

First, we need to write a figure class for the new diamond shape, which extends from the existing class `Figure`. Then we need to override the `paintFigure` method as shown below:

```
public class DiamondFigure extends Figure
{
    public DiamondFigure(Rectangle rect)
    {
        setBounds(rect);
    }

    protected void paintFigure(Graphics g)
    {
        Rectangle r = getParent().getBounds().getCopy();
        setBounds(r);

        PointList points = new PointList();
        points.addPoint(new Point(r.x + r.width / 2, r.y));
        points.addPoint(new Point(r.x + r.width, r.y + r.height / 2));
        points.addPoint(new Point(r.x + r.width / 2, r.y + r.height));
        points.addPoint(new Point(r.x, r.y + r.height / 2));

        g.fillPolygon(points);
        g.drawPolygon(points);
    }
}
```

Above code segment implements a figure that draws a diamond. You can add this code into `NodeFigure` class as other figure classes for different shapes do (`RectangleFigure, EllipseFigure` and `TriangleFigure` classes).

Now we need to assign a name string to our new shape by adding it into the static array in `NodeModel` as shown below:

```
public static String[] shapes = {
    "Rectangle",
    "Ellipse",
    "Triangle",
    "Diamond"};
```

`NodeFigure` class has an `updateShape` method. This method decides which figure will be drawn. So we need to add our diamond figure into this method. In addition, we must associate our newly added name string to our new diamond figure.

```
public void updateShape(String s)
{
    this.shape = s;
    this.removeAll();
```

```
    if (shape.equals(NodeModel.shapes[0]))
    {
        add(new RectangleFigure(getBounds()));
    }
    else if (shape.equals(NodeModel.shapes[1]))
    {
        add(new EllipseFigure(getBounds()));
    }
    else if (shape.equals(NodeModel.shapes[2]))
    {
        add(new TriangleFigure(getBounds()));
    }
    else if (shape.equals(NodeModel.shapes[3]))
    {
        add(new DiamondFigure(getBounds()));
    }

    add(label);
}
```

Here, we must be careful about the index of the newly added shape name `"Diamond"`. It is in the 4th place of the static array, so if shape's type is the same with the element in the 4th place of static array (`NodeModel.shapes[3]`), this means that shape is a diamond and we create our new diamond figure.

By updating this method, we have completed addition of a new shape to Chisio nodes. See **Figure 4** for examples of the new node shape.

Figure 4 Sample UIs for two diamond shaped nodes.

Adding new attributes

Each graph object (simple node, compound node or edge) in Chisio has several attributes. Existing attributes may be sufficient for drawings of many types of graphs. However specific applications are most likely to have other attributes. For example, one may want to maintain a dynamic size for the arrow heads of directed edges.

We first need to update the `EdgeModel` class to handle such a new attribute for edges. Specifically we need to add a field named `arrowSize` (say, with type `int`). In addition, a default value for this new field is needed (`DEFAULT_ARROW_SIZE`). Default values are defined in the "Class Variables" section of each class.

Furthermore, a constant variable is needed for firing the property change. When we change the arrow size, this change is to be propagated to the corresponding edit-part. So, an identifier is defined in "Class Constants" section (`P_ARROW_SIZE`).

As usual, operation setter and getter methods are written for the newly created field. But in here, setter method fires the property change mentioned earlier.

```
public class EdgeModel extends GraphObject
{
    private NodeModel sourceNode;
    private NodeModel targetNode;
    protected String style;
    protected String arrow;
```

```
    protected int width;
    protected List bendpoints;
    protected int arrowSize;

    /**
     * Constructor
     */
    public EdgeModel()
    {
        super();
        this.text = DEFAULT_TEXT;
        this.textFont = DEFAULT_TEXT_FONT;
        this.textColor = DEFAULT_TEXT_COLOR;
        this.color = DEFAULT_COLOR;
        this.style = DEFAULT_STYLE;
        this.arrow = DEFAULT_ARROW;
        this.width = DEFAULT_WIDTH;
        this.bendpoints = new ArrayList();
        this.arrowSize = DEFAULT_ARROW_SIZE;
    }

    ...

    public int getArrowSize()
    {
        return arrowSize;
    }

    public void setArrowSize(int arrowSize)
    {
        this.arrowSize = arrowSize;
        firePropertyChange(P_ARROW_SIZE, null, arrow_size);
    }

    ...

// ------------------------------------------------------------------------
// Section: Class Variables
// ------------------------------------------------------------------------
    public static String DEFAULT_TEXT = "";

    public static Font DEFAULT_TEXT_FONT =
        new Font(null, new FontData("Arial", 8, SWT.NORMAL));

    ...

    public static int DEFAULT_WIDTH = 1;

    public static int DEFAULT_ARROW_SIZE = 1;

// ------------------------------------------------------------------------
// Section: Class Constants
// ------------------------------------------------------------------------
    public static final String P_STYLE = "_style";
    public static final String P_ARROW = "_arrow";
    public static final String P_WIDTH = "_width";
    public static final String P_BENDPOINT = "_bendpoint";
    public static final String P_ARROW_SIZE = "_arrowSize";
}
```

The model including the new attribute is now ready. Let's modify the figure part accordingly now. For this, we must add a new field and an update method to set this field and change the UI. We might also need to update the `paintFigure` method, which draws the UI.

```
public class EdgeFigure extends PolylineConnection
{
    ...

    boolean highlight;
    Color highlightColor;
    int arrowSize;

    /**
     * Constructor
     */
    public EdgeFigure(String text,
        Font textFont,
        Color textColor,
        Color color,
        String style,
        String arrow,
        int width,
        Color highlightColor,
        boolean highlight,
        int arrowSize)
    {
        super();

        ...

        updateWidth(width);
        updateHighlightColor(highlightColor);
        updateArrowSize(arrowSize);
    }

    ...

    public void updateArrowSize(int newArrowSize)
    {
        // Do the operations for setting the arrow size
    }

    ...
}
```

Now that we have updated the model and figure parts, we must connect them to each other. We need to update the edit-part for propagating the changes in the model to the figure (UI). The edit-part of an edge is implemented in class `ChsEdgeEditPart`. We must update `createFigure` and `propertyChange` methods in this class.

```
protected IFigure createFigure()
{
    EdgeModel model = getEdgeModel();
    EdgeFigure eFigure = new EdgeFigure(model.getText(),
        model.getTextFont(),
        model.getTextColor(),
        model.getColor(),
        model.getStyle(),
```

```
            model.getArrow(),
            model.getWidth(),
            model.getHighlightColor(),
            model.isHighlight(),
            model.getArrowSize());

        eFigure.updateHighlight(
            (HighlightLayer) getLayer(HighlightLayer.HIGHLIGHT_LAYER),
            getEdgeModel().isHighlight());

        ...
    }

    public void propertyChange(PropertyChangeEvent evt)
    {
        if (evt.getPropertyName().equals(EdgeModel.P_TEXT))
        {
            ((EdgeFigure)figure).updateText((String) evt.getNewValue());
        }

        ...

        else if (evt.getPropertyName().equals(EdgeModel.P_HIGHLIGHT))
        {
            ((EdgeFigure)figure).updateHighlight(
                (Layer) getLayer(HighlightLayer.HIGHLIGHT_LAYER));
        }
        else if (evt.getPropertyName().equals(EdgeModel.P_HIGHLIGHTCOLOR))
        {
            ((EdgeFigure)figure).
                updateHighlightColor((Color) evt.getNewValue());
        }
        else if (evt.getPropertyName().equals(EdgeModel.P_ARROW_SIZE))
        {
            ((EdgeFigure)figure).updateArrowSize((Integer) evt.getNewValue());
        }
    }
```

Thus we have completed adding a new attribute into Chisio edges. In the next part, we will make the necessary changes to expose this new attribute to the users through the edge inspector window.

Modifying inspector windows

The implementation of object property inspector windows in Chisio are based on a class named `Inspector` and specific inspectors extend this class (e.g. `EdgeInspector`). If you want to add a new item to the edge inspector, first you must add it into `EdgeInspector` class as follows. We will just change the constructor of the class.

```
private EdgeInspector(GraphObject model, String title, ChisioMain main)
{
    super(model, title, main);

    TableItem item = addRow(table, "Name");
    item.setText(1, model.getText());

    ...

    item = addRow(table, "Width");
    item.setText(1, "" + ((EdgeModel) model).getWidth());
```

```
    item = addRow(table, "Arrow Size");
    item.setText(1, "" + ((EdgeModel) model).getArrowSize());

    createContents(shell);

    ...
}
```

Here we add a new row with attribute name "Arrow Size", whose value is taken from the corresponding model object. Now, we need to update `createContents` method of `Inspector` class to add interactivity to our newly added attribute. We already had five types of objects in our edge inspector tables: *Combo*, *Text*, *Number*, *Color* and *Font*. Our new attribute's value is a number. So we need to add it into the if-statement, which is tagged with "NUMBER".

```
public void createContents(final Shell shell)
{
    // Create an editor object to use for text editing
    final TableEditor editor = new TableEditor(table);
    editor.horizontalAlignment = SWT.LEFT;
    editor.grabHorizontal = true;

    // Use a selection listener to get seleceted row
    table.addSelectionListener(new SelectionAdapter()
    {
        public void widgetSelected(SelectionEvent event)
        {
            // Dispose any existing editor
            Control old = editor.getEditor();

            ...

            if (item != null)
            {
                // COMBO
                if (item.getText().equals("Style")
                    || item.getText().equals("Arrow")
                    || item.getText().equals("Shape"))
                {
                    ...
                }
                // TEXT
                else if (item.getText().equals("Name"))
                {
                    ...
                }
                // NUMBER
                else if (item.getText().equals("Margin")
                    || item.getText().equals("Cluster ID")
                    || item.getText().equals("Width")
                    || item.getText().equals("Arrow Size"))
                {
                    ...
                }
                // COLOR
                else if (item.getText().equals("Border Color")
                    || item.getText().equals("Color")
                    || item.getText().equals("Highlight Color"))
                {
                    ...
```

```
                }
                // FONT
                else if (item.getText().equals("Text Font"))
                {
                    ...
                }

                table.setSelection(-1);
            }
        }
    });

    ...
}
```

We have added new attribute into our inspector window (**Figure 5**). When "OK" button is pressed, values are transferred to the model. So we need to update the listener for the "OK" button to add the newly added arrow size attribute as below.

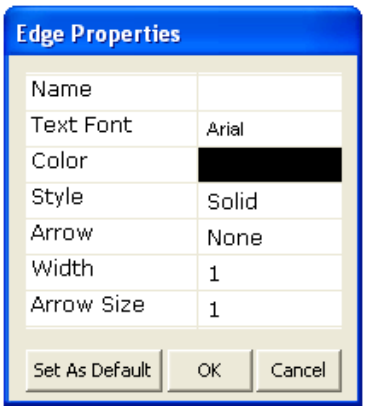

Figure 5 Sample screenshot of the Edge Inspector after addition of the new attribute "Arrow Size".

```
okButton.addSelectionListener(new SelectionAdapter()
{
    public void widgetSelected(final SelectionEvent e)
    {
        TableItem[] items = table.getItems();

        for (TableItem item : items)
        {
            if (item.getText().equals("Name"))
            {
                model.setText(item.getText(1));
            }

            ...

            else if (item.getText().equals("Margin"))
            {
                new ChangeMarginAction((CompoundModel)model,
                    Integer.parseInt(item.getText(1))).run();
            }
            else if (item.getText().equals("Arrow Size"))
            {
                ((EdgeModel) model).
                    setArrowSize(Integer.parseInt(item.getText(1)));
            }
        }
```

```
        shell.close();
    }
});
```

Furthermore, we need to update `setAsDefault` method in `EdgeInspector` class to change the default arrow size with this new value.

```
public void setAsDefault()
{
    TableItem[] items = table.getItems();

    for (TableItem item : items)
    {
        if (item.getText().equals("Name"))
        {
            EdgeModel.DEFAULT_TEXT = item.getText(1);
        }

        ...

        else if (item.getText().equals("Width"))
        {
            EdgeModel.DEFAULT_WIDTH = Integer.parseInt(item.getText(1));
        }
        else if (item.getText().equals("Arrow Size"))
        {
            EdgeModel.DEFAULT_ARROW_SIZE = Integer.parseInt(item.getText(1));
        }
    }
}
```

Viola! We have now completed adding a new item into an inspector.

Changing defaults for nodes and edges

There are default values for the creation of simple and compound nodes and edges. These values can be changed easily by updating a few lines.

Defaults for nodes are kept in the `NodeModel` class in the "Class Variables" section. By changing these variables, we can change the defaults for nodes.

```
// ----------------------------------------------------------------------------
// Section: Class Variables
// ----------------------------------------------------------------------------
    public static Dimension DEFAULT_NODE_SIZE = new Dimension(40, 40);

    public static String DEFAULT_TEXT = "Node";

    public static Font DEFAULT_TEXT_FONT =
        new Font(null, new FontData("Arial", 8, SWT.NORMAL));

    public static Color DEFAULT_TEXT_COLOR = ColorConstants.black;

    public static Color DEFAULT_COLOR = new Color(null, 14, 112, 130);

    public static Color DEFAULT_BORDER_COLOR = new Color(null, 14, 112, 130);

    public static String DEFAULT_SHAPE = shapes[0];
```

```
    public static int DEFAULT_CLUSTER_ID = 0;
```

Compound node and edge default values may be changed similarly under the "Class Variables" section of `CompoundModel` and `EdgeModel` classes, respectively.

File Operations

Persistence of newly added attributes requires special attention. Below you will find what you need to do so these attributes can be saved and loaded back properly on disk.

Integrating new attributes into save operation

We have added the "arrow size" attribute for edges earlier on. Now, we will add support for this attribute in the save operation.

There is a `GraphMLWriter` class for saving our graphs as a ".graphml" file. We need to update this class.

First, we must define a `KeyType` for arrow size attribute (`arrowSizeKey`). Also we need to add code into `writeXMLFile` and `createEdge` methods as shown below. If we had added a new parameter for nodes, then we should have changed `createNode` method as well.

```
public class GraphMLWriter extends XMLWriter
{
    HashMap hashMap = new HashMap();

    GraphmlType newGraphml;

    KeyType xKey;

    ...

    KeyType highlightColorKey;

    KeyType arrowSizeKey;

    public Object writeXMLFile(CompoundModel root)
    {
        // create a new graphml file
        GraphmlDocument newGraphmlDoc = GraphmlDocument.Factory.newInstance();
        newGraphml = newGraphmlDoc.addNewGraphml();

        // define the keys that will be used in xml file
        xKey = newGraphml.addNewKey();
        xKey.setId("x");
        xKey.setFor(KeyForType.NODE);
        xKey.setAttrName("x");
        xKey.setAttrType(KeyTypeType.INT);

        ...

        highlightColorKey = newGraphml.addNewKey();
        highlightColorKey.setId("highlightColor");
        highlightColorKey.setFor(KeyForType.ALL);
        highlightColorKey.setAttrName("highlightColor");
        highlightColorKey.setAttrType(KeyTypeType.STRING);

        arrowSizeKey = newGraphml.addNewKey();
        arrowSizeKey.setId("arrowSize");
```

```
        arrowSizeKey.setFor(KeyForType.NODE);
        arrowSizeKey.setAttrName("arrowSize");
        arrowSizeKey.setAttrType(KeyTypeType.INT);

        // create a new graph with our root node recursively
        GraphType newGraph = newGraphml.addNewGraph();
        createTree(newGraph, root, "");

        return newGraphmlDoc;
    }

    public void createEdge(EdgeType newEdge, EdgeModel model)
    {
        // write edge's properties into file
        DataType colorData = newEdge.addNewData();
        colorData.setKey(colorKey.getId());
        RGB rgb = model.getColor().getRGB();
        colorData.set(XmlString.Factory.newValue(rgb.red + " " +
            rgb.green + " " + rgb.blue));

        ...

        if (model.isHighlight())
        {
            DataType highlightColorData = newEdge.addNewData();
            highlightColorData.setKey(highlightColorKey.getId());
            rgb = model.getHighlightColor().getRGB();
            highlightColorData.set(XmlString.Factory.
                newValue(rgb.red + " " + rgb.green + " " + rgb.blue));
        }

        DataType arrowSizeData = newEdge.addNewData();
        arrowSizeData.setKey(arrowSizeKey.getId());
        arrowSizeData.set(XmlString.Factory.
            newValue("" + model.getArrowSize()));
    }
}
```

That's all. We now have support for our new attribute on save.

Integrating new attributes into load operation

Now that we know how to save graphs with newly introduced attribute "arrow size", let's now integrate this attribute into the load operation. For this purpose, we need to update `readEdge` method in `GraphMLReader` class as shown below. If we had added a new parameter for nodes, then we should have updated the `readNode` method as well.

```
public void readEdge(EdgeType edge, EdgeModel model)
{
    int width = -1;
    String style = null;

    ...

    int arrowSize = -1;

    // read edge's properties
    DataType[] datas = edge.getDataArray();

    for (int i = 0; i < datas.length; i++)
```

```
    {
        DataType data = datas[i];

        if (data.getKey().equalsIgnoreCase("color"))
        {
            color = data.newCursor().getTextValue();
        }

        ...

        else if (data.getKey().equalsIgnoreCase("highlightColor"))
        {
            highlightColor = data.newCursor().getTextValue();
        }
        else if (data.getKey().equalsIgnoreCase("arrowSize"))
        {
            arrowSize = Integer.parseInt(data.newCursor().getTextValue());
        }
    }

    ...

    if (highlightColor != null)
    {
        String[] rgb = highlightColor.split(" ");
        int r = Integer.parseInt(rgb[0]);
        int g = Integer.parseInt(rgb[1]);
        int b = Integer.parseInt(rgb[2]);
        model.setHighlightColor(new Color(null, r, g, b));
        model.setHighlight(true);
    }

    if (arrowSize > 0)
    {
        model.setArrowSize(arrowSize);
    }
}
```

By reading the `arrowSize` field in ".graphml" files, we have added the support for loading newly created attribute "arrow size".

Reading other file formats

Chisio uses `GraphML` file format for persistent storage of graphs. There are two classes `GraphMLReader` and `GraphMLWriter` for load and save operations, which are extended from abstract classes `XMLReader` and `XMLWriter`.

If you would like to support other file formats such as GXL, GML or your own XML-based format, you must extend `XMLReader` and `XMLWriter` classes accordingly. Suppose we want to support ".gxl" files instead.

```
public class GXLReader extends XMLReader
{
    public CompoundModel readXMLFile(File xmlFile)
    {
        // parse GXL file in here and return the read root graph
    }
}

public class GXLWriter extends XMLWriter
```

```
{
    public Object writeXMLFile(CompoundModel root)
    {
        // create GXL file from root graph return the file content
    }
}
```

After writing new classes named `GXLReader` and `GXLWriter` similar to their ".graphml" counterpart, we need to change `LoadAction` and `SaveAction` classes as below.

```
public class LoadAction extends Action
{
    ...

    public void run()
    {
        ...

        File xmlfile = new File(filename);

        XMLReader reader = new GXLReader();
        CompoundModel root = reader.readXMLFile(xmlfile);

        ...
    }
}

public class SaveAction extends Action
{
    ...

    public void run()
    {
        ...

        BufferedWriter xmlFile =
            new BufferedWriter(new FileWriter(fileName));

        XMLWriter writer = new GXLWriter();
        xmlFile.write(writer.writeXMLFile(root).toString());
        xmlFile.close();

        ...
    }
}
```

In the `run` method of `LoadAction` class, we simply change `GraphMLReader` to `GXLReader` and in the run method of `SaveAction` class, change `GraphMLWriter` to `GXLWriter`. That's all.

Menu Operations

Each operation in the toolbar menu, pop-up menus or the main menu has an associated `Action`. So, if you would like to add a new menu item, you must first create an associated action for it.

For example, let's suppose we want to add a new operation for finding neighbors of a selected node and highlight them. We can write a new action for this operation as follows. Icon for this action is set in the constructor method.

```
import org.eclipse.jface.action.Action;

public class HighlightNeighborsAction extends Action
{
    ChisioMain main;

    /**
     * Constructor
     */
    public HighlightNeighborsAction(ChisioMain main)
    {
        super("Highlight Neighbors");
        this.setToolTipText("Highlight Neighbors");
        setImageDescriptor(ImageDescriptor.createFromFile(
            ChisioMain.class,
            "icon/highlight-neighbors.gif"));
        this.main = main;
    }

    public void run()
    {
        ScrollingGraphicalViewer viewer = main.getViewer();
        // Iterates selected objects; for each selected objects, highlights them
        Iterator selectedObjects =
            ((IStructuredSelection) viewer.getSelection()).iterator();

        if (selectedObjects.hasNext())
        {
            Object model = ((EditPart)selectedObjects.next()).getModel();

            if (model instanceof NodeModel)
            {
                Iterator<NodeModel> neighbors =
                    ((NodeModel) model).getNeighborsList().iterator();

                while (neighbors.hasNext())
                {
                    NodeModel next = neighbors.next();
                    next.setHighlightColor(main.higlightColor);
                    next.setHighlight(true);
                }
            }
        }
    }
}
```

Now, we can use this action to create a new menu item.

New main menu item

We must update `createBarMenu` method in `TopMenuBar` class for adding this action to the main menu. Suppose the new operation is to be added under the "View" menu.

```
public static MenuManager createBarMenu(ChisioMain main)
{
    chisio = main;

    MenuManager menuBar = new MenuManager("");
    MenuManager fileMenu = new MenuManager("&File");
    MenuManager editMenu = new MenuManager("&Edit");
    MenuManager viewMenu = new MenuManager("&View");
```

```
    ...

    viewMenu.add(new Separator());
    viewMenu.add(new HighlightSelectedAction(chisio));
    viewMenu.add(new HighlightNeighborsAction(chisio));
    viewMenu.add(new RemoveHighlightFromSelectedAction(chisio));
    viewMenu.add(new RemoveHighlightsAction(chisio));

    ...
}
```

New pop menu item

We must update `createActions` method in `PopupManager` class for adding this new operation to popup menu. This is an action for nodes/compound nodes. So we must add it to `NODE-COMPOUND POPUP` part of the code.

```
public void createActions(IMenuManager manager)
{
    EditPart ep = main.getViewer().findObjectAt(clickLocation);

    if (ep instanceof RootEditPart)
    {
        // GRAPH POPUP
        manager.add(new ZoomAction(main, 1, clickLocation));
        manager.add(new ZoomAction(main, -1, clickLocation));
        manager.add(new RemoveHighlightsAction(main));
        manager.add(new LayoutInspectorAction(main));
        manager.add(new InspectorAction(main, true));
        main.getViewer().select(ep);
    }
    else if (ep instanceof NodeEditPart)
    {
        // NODE-COMPOUND POPUP
        manager.add(new CreateCompoundFromSelectedAction(main));
        manager.add(new HighlightSelectedAction(main));
        manager.add(new HighlightNeighborsAction(main));
        manager.add(new RemoveHighlightFromSelectedAction(main));
        manager.add(new DeleteAction(main.getViewer()));
        manager.add(new InspectorAction(main, false));
    }
    else if (ep instanceof ChsEdgeEditPart)
    {
        // EDGE POPUP
        manager.add(new HighlightSelectedAction(main));
        manager.add(new RemoveHighlightFromSelectedAction(main));
        manager.add(new DeleteAction(main.getViewer()));
        manager.add(new InspectorAction(main, false));
    }
}
```

New toolbar item

We must update `ToolbarManager` class for adding this operation to the toolbar menu. The icon for this item is already set in the constructor of `HighlightNeighborsAction` class.

```
public ToolbarManager(int style, ChisioMain main)
{
    super(style);

    add(new NewAction("New", main));
```

```
    add(new LoadAction(main));
    add(new SaveAction(main));

    ...

    add(new HighlightNeighborsAction(main));

    ...

    add(new SugiyamaLayoutAction(main));
    add(new Separator());

    update(true);
}
```

Chapter 3

Adding New Layout Operations into Chisio

In this chapter, we describe how one can integrate a new layout operation into the existing layout architecture of Chisio.

Chisio currently supports several layout styles from the basic spring embedder to hierarchical (Sugiyama) layout to compound spring embedder to circular layout. For details on these layout operations, please refer to Chisio User's Guide.

Chisio layout architecture

The basis in Chisio layout is constructed through an abstract layout class and an associated l-structure to represent compound graphs named `AbstractLayout` and `LGraphManager/LGraph/LNode/LEdge`, respectively. Here an `LGraphManager` maintains and manages a list of `LGraph` instances, which in turn maintains a list of `LNode`'s and `LEdge`'s. Notice here that similar to the Chisio structure, the top-most level is composed of a *root node* and a *root graph* (root node's child graph) composing the actual content of the graph structure. In other words, the root node and the root graph are there even if the graph is empty!

The `AbstractLayout` class converts the given Chisio model into a generic l-structure used only for layout purposes that is destroyed at the end of layout. Individual layout algorithms extend the `AbstractLayout` class and run on this l-structure. When the layout is finished, the geometry information of the l-level is transferred back to Chisio model.

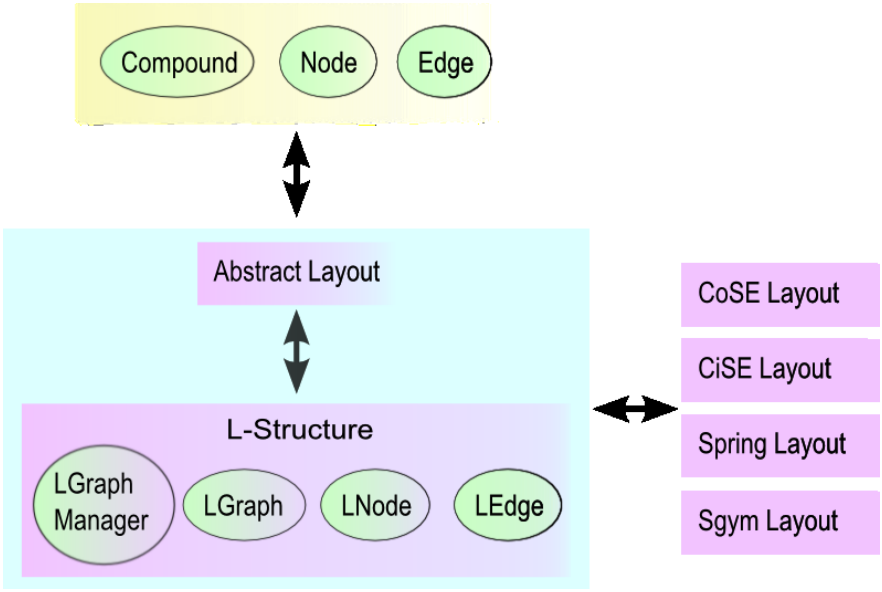

Figure 6 Layout architecture in Chisio

Each layout operation runs on a separate thread for more robust animation as well as for allowing cancellation.

In addition, all layout operations can be run without any need for Chisio window (i.e. graphical user interface). This facilitates running layout in batch mode, especially useful for testing purposes.

Adding new layout algorithms

Currently, Chisio hosts the following layout styles:

- CoSE: Compound graph layout algorithm based on a spring embedder
- Cluster Layout: Members of each cluster is grouped and laid out together
- CiSE: Circular graph layout algorithm based on a spring embedder
- Circular Layout: AVSDF circular layout
- Spring Layout: Basic spring embedder
- Sugiyama Layout: A hierarchical layout style based on Sugiyama algorithm

Detailed information about these algorithms and our implementation, please refer to the User's Guide. All these layouts extend `AbstractLayout` class and its abstract methods are overridden to implement the corresponding layout algorithm. If you would like to add a new layout algorithm, you must write a new class extending `AbstractLayout` as well.

For example, let's suppose we want to add random layout, which randomly scatters the nodes, as a new style.

```
public class RandomLayout extends AbstractLayout
{
    public RandomLayout(CompoundModel rootModel)
    {
        super(rootModel);
    }

    public void layout()
    {
        this.positionNodesRandomly();
    }
}
```

In here, `layout` method, which is abstract in the base class is to include the actual layout algorithm. If we need some initialization for layout, we can override the `initialize` method, which is also in `AbstractLayout`.

In case we need layout-style specific nodes and edges for our layout algorithm, we should extend `LNode` and `LEdge` classes, respectively, and override the methods `createNewLNode` and `createNewLEdge` in `AbstractLayout` as shown below.

```
import org.eclipse.draw2d.geometry.*;

public class RandomNode extends LNode
{
    public RandomNode(LGraphManager gm)
    {
        super(gm);
    }

    public RandomNode(LGraphManager gm, Point loc, Dimension size)
    {
        super(gm, loc, size);
    }

    ...

}

public class RandomLayout extends AbstractLayout
{
    public RandomLayout(CompoundModel rootModel)
    {
        super(rootModel);
    }

    public LNode createNewLNode(LGraphManager gm, NodeModel model)
    {
        return new RandomNode(gm);
    }

    public LNode createNewLNode(LGraphManager gm,
        NodeModel model,
        Point loc,
        Dimension size)
    {
        return new RandomNode(gm, loc, size);
    }

    public void layout()
```

```
    {
        this.positionNodesRandomly();
    }
}
```

New layout options

`LayoutOptionsPack` is a class for maintaining the layout parameters that can be customized by the user. This class has an inner class for each layout style to handle parameters belonging to that particular layout style. Hence, we should add an inner class for our new layout style (i.e. random layout).

```
public class LayoutOptionsPack implements Serializable
{
    private static LayoutOptionsPack instance;

    private General general;
    private CoSE coSE;
    private Cluster cluster;
    private CiSE ciSE;
    private AVSDF avsdf;
    private Spring spring;
    private Sgym sgym;
    private Random random;

    public class General
    {
        ...
    }

    public class CoSE
    {
        ...
    }

    ...

    public class Random
    {
        // our parameters and setter, getter methods
    }

    private LayoutOptionsPack()
    {
        this.general = new General();
        this.coSE = new CoSE();
        this.cluster = new Cluster();
        this.ciSE = new CiSE();
        this.avsdf = new AVSDF();
        this.spring = new Spring();
        this.sgym = new Sgym();
        this.random = new Random();

        setDefaultLayoutProperties();
    }

    public void setDefaultLayoutProperties()
    {
        general.setAnimationPeriod(50);
        general.setAnimationDuringLayout(
            AbstractLayout.DEFAULT_ANIMATION_DURING_LAYOUT);
```

```
        general.setAnimationOnLayout(
            AbstractLayout.DEFAULT_ANIMATION_ON_LAYOUT);

        ...

        random.set...
        random.set...
    }

    public General getGeneral()
    {
        return general;
    }

    ...

    public Random getRandom()
    {
        return random;
    }
}
```

We have updated layout options pack which gives us the ability to parameterize our layouts. We can easily change the layout properties window by adding a new tab for our new style as well. Our new layout style will use the values in layout options pack, which will be set through a new tab in the layout properties window as described below.

```
public class LayoutInspector extends Dialog
{
    ...

    protected void createContents()
    {
        ...

        // Create the UI for new layout. Buttons, sliders, checkboxes etc.

        ...
    }

    public void storeValuesToOptionsPack()
    {
        LayoutOptionsPack lop = LayoutOptionsPack.getInstance();

        // General
        lop.getGeneral().setAnimationPeriod(animationPeriod.getSelection());
        lop.getGeneral().setAnimationOnLayout(
            animateOnLayoutButton.getSelection());

        ...

        // Random
        lop.getRandom().set..
        lop.getRandom().set..
    }

    public void setInitialValues()
    {
        LayoutOptionsPack lop = LayoutOptionsPack.getInstance();
```

```
        // General
        animationPeriod.setSelection(lop.getGeneral().getAnimationPeriod());
        animateDuringLayoutButton.setSelection(
            lop.getGeneral().isAnimationDuringLayout());

        ...

        // Random

        // Take values from UI and set into layout options pack (lop)
    }

    public void setDefaultLayoutProperties(int select)
    {
        if (select == 0)
        {
            // General
            animateDuringLayoutButton.setSelection(
                AbstractLayout.DEFAULT_ANIMATION_DURING_LAYOUT);
            animationPeriod.setSelection(50);

            ...
        }

        ...

        else if (select == 7)
        {
            // Random

            // Fill the UI with the default values of layout
        }
    }
}
```

Thus, we have added a new tab for our new layout algorithm to set its associated exposed parameters.

Changing defaults for layouts

The default values for both exposed and non-exposed parameters of a layout style may be modified. For exposed parameters these refer to the defaults in Layout Properties window. For non-exposed ones, the only way to change a layout parameter is by updating the default values in the associated section of the corresponding layout class.

For example, suppose we want to change the default vertical spacing (spacing between levels) for our hierarchical (Sugiyama) layout. Then in the `SgymLayout` class, simply go to the "Class Constants" section and change the default value of the constant `DEFAULT_VERTICAL_SPACING` to its new value as shown below.

```
// ------------------------------------------------------------------------
// Section: Class Constants
// ------------------------------------------------------------------------
    public final static int DEFAULT_HORIZONTAL_SPACING = 100;

    public final static int DEFAULT_VERTICAL_SPACING = 120; // was 80

    public final static boolean DEFAULT_VERTICAL = true;
```

Performance analysis

The performance of a particular layout algorithm could be measured by looking at certain properties (e.g. number of crossings, total area, and execution time) of a laid out drawing. For this purpose, you may use the class `org.gvt.layout.PerformanceAnalyser`.

The class `org.gvt.util.RandomGraphCreator` may be used to create random graphs for performance analysis. You may refer to the User's Guide for details on random graph creation. In addition, a good number of connected bi-planar subset of Rome graphs (http://www.dia.uniroma3.it/~gdt/gdt4/test_suite.php) have been imported and are available for batch or interactive use in GraphML format.

Chapter 4

Third-Party Software License Agreements

In this chapter, we list any third-party software license agreements.

Chisio makes use of Eclipse Graph Editing Framework (GEF) version 3.1 distributed under license Eclipse Public License – v 1.0, for drawing and editing pathways. Silk icons, which are distributed under Creative Commons Attribution 2.5 License, have been used to design icons for the toolbar.

Eclipse Public License – v 1.0

THE ACCOMPANYING PROGRAM IS PROVIDED UNDER THE TERMS OF THIS ECLIPSE PUBLIC LICENSE ("AGREEMENT"). ANY USE, REPRODUCTION OR DISTRIBUTION OF THE PROGRAM CONSTITUTES RECIPIENT'S ACCEPTANCE OF THIS AGREEMENT.

1. DEFINITIONS

"Contribution" means:

a) in the case of the initial Contributor, the initial code and documentation distributed under this Agreement, and
b) in the case of each subsequent Contributor:

i)changes to the Program, and

ii)additions to the Program;

where such changes and/or additions to the Program originate from and are distributed by that particular Contributor. A Contribution 'originates' from a Contributor if it was added to the Program by such Contributor itself or anyone acting on such Contributor's behalf. Contributions do not include additions to the Program which: (i) are separate modules of software distributed in conjunction with the Program under their own license agreement, and (ii) are not derivative works of the Program.

"Contributor" means any person or entity that distributes the Program.

"Licensed Patents " mean patent claims licensable by a Contributor which are necessarily infringed by the use or sale of its Contribution alone or when combined with the Program.

"Program" means the Contributions distributed in accordance with this Agreement.

"Recipient" means anyone who receives the Program under this Agreement, including all Contributors.

2. GRANT OF RIGHTS

a) Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free copyright license to reproduce, prepare derivative works of, publicly display, publicly perform, distribute and sublicense the Contribution of such Contributor, if any, and such derivative works, in source code and object code form.

b) Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free patent license under Licensed Patents to make, use, sell, offer to sell, import and otherwise transfer the Contribution of such Contributor, if any, in source code and object code form. This patent license shall apply to the combination of the Contribution and the Program if, at the time the Contribution is added by the Contributor, such addition of the Contribution causes such combination to be covered by the Licensed Patents. The patent license shall not apply to any other combinations which include the Contribution. No hardware per se is licensed hereunder.

c) Recipient understands that although each Contributor grants the licenses to its Contributions set forth herein, no assurances are provided by any Contributor that the Program does not infringe the patent or other intellectual property rights of any other entity. Each Contributor disclaims any liability to Recipient for claims brought by any other entity based on infringement of intellectual property rights or otherwise. As a condition to exercising the rights and licenses granted hereunder, each Recipient hereby assumes sole responsibility to secure any other intellectual property rights needed, if any. For example, if a third party patent license is required to allow Recipient to distribute the Program, it is Recipient’s responsibility to acquire that license before distributing the Program.

d) Each Contributor represents that to its knowledge it has sufficient copyright rights in its Contribution, if any, to grant the copyright license set forth in this Agreement.

3. REQUIREMENTS

A Contributor may choose to distribute the Program in object code form under its own license agreement, provided that:

a) it complies with the terms and conditions of this Agreement; and

b) its license agreement:

i) effectively disclaims on behalf of all Contributors all warranties and conditions, express and implied, including warranties or conditions of title and non-infringement, and implied warranties or conditions of merchantability and fitness for a particular purpose;

ii) effectively excludes on behalf of all Contributors all liability for damages, including direct, indirect, special, incidental and consequential damages, such as lost profits;

iii) states that any provisions which differ from this Agreement are offered by that Contributor alone and not by any other party; and

iv) states that source code for the Program is available from such Contributor, and informs licensees how to obtain it in a reasonable manner on or through a medium customarily used for software exchange.

When the Program is made available in source code form:

a) it must be made available under this Agreement; and

b) a copy of this Agreement must be included with each copy of the Program.

Contributors may not remove or alter any copyright notices contained within the Program.

Each Contributor must identify itself as the originator of its Contribution, if any, in a manner that reasonably allows subsequent Recipients to identify the originator of the Contribution.

4. COMMERCIAL DISTRIBUTION

Commercial distributors of software may accept certain responsibilities with respect to end users, business partners and the like. While this license is intended to facilitate the commercial use of the Program, the Contributor who includes the Program in a commercial product offering should do so in a manner which does not create potential liability for other Contributors. Therefore, if a Contributor includes the Program in a commercial product offering, such Contributor ("Commercial Contributor") hereby agrees to defend and indemnify every other Contributor ("Indemnified Contributor") against any losses, damages and costs (collectively "Losses") arising from claims, lawsuits and other legal actions brought by a third party against the Indemnified Contributor to the extent caused by the acts or omissions of such Commercial Contributor in connection with its distribution of the Program in a commercial product offering. The obligations in this section do not apply to any claims or Losses relating to any actual or alleged intellectual property infringement. In order to qualify, an Indemnified Contributor must: a) promptly notify the Commercial Contributor in writing of such claim, and b) allow the Commercial Contributor to control, and cooperate with the Commercial Contributor in, the defense and any related settlement negotiations. The Indemnified Contributor may participate in any such claim at its own expense.

For example, a Contributor might include the Program in a commercial product offering, Product X. That Contributor is then a Commercial Contributor. If that Commercial Contributor then makes performance claims, or offers warranties related to Product X, those performance claims and warranties are such Commercial Contributor's responsibility alone. Under this section, the Commercial Contributor would have to defend claims against the other Contributors related to those performance claims and warranties, and if a court requires any other Contributor to pay any damages as a result, the Commercial Contributor must pay those damages.

5. NO WARRANTY

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, THE PROGRAM IS PROVIDED ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED INCLUDING, WITHOUT LIMITATION, ANY WARRANTIES OR CONDITIONS OF TITLE, NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Each Recipient is solely responsible for determining the appropriateness of using and distributing the Program and assumes all risks associated with its exercise of rights under this Agreement , including but not limited to the risks and costs of program errors, compliance with applicable laws, damage to or loss of data, programs or equipment, and unavailability or interruption of operations.

6. DISCLAIMER OF LIABILITY

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, NEITHER RECIPIENT NOR ANY CONTRIBUTORS SHALL HAVE ANY LIABILITY FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING WITHOUT LIMITATION LOST PROFITS), HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OR DISTRIBUTION OF THE PROGRAM OR THE EXERCISE OF ANY RIGHTS GRANTED HEREUNDER, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. GENERAL

If any provision of this Agreement is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this Agreement, and without further action by the parties hereto, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

If Recipient institutes patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Program itself (excluding combinations of the Program with other software or hardware) infringes such Recipient's patent(s), then such Recipient's rights granted under Section 2(b) shall terminate as of the date such litigation is filed.

All Recipient's rights under this Agreement shall terminate if it fails to comply with any of the material terms or conditions of this Agreement and does not cure such failure in a reasonable period of time after becoming aware of such noncompliance. If all Recipient's rights under this Agreement terminate, Recipient agrees to cease use and distribution of the Program as soon as reasonably practicable. However, Recipient's obligations under this Agreement and any licenses granted by Recipient relating to the Program shall continue and survive.

Everyone is permitted to copy and distribute copies of this Agreement, but in order to avoid inconsistency the Agreement is copyrighted and may only be modified in the following manner. The Agreement Steward reserves the right to publish new versions (including revisions) of this Agreement from time to time. No one other than the Agreement Steward has the right to modify this Agreement. The Eclipse Foundation is the initial Agreement Steward. The Eclipse Foundation may assign the responsibility to serve as the Agreement Steward to a suitable separate entity. Each new version of the Agreement will be given a distinguishing version number. The Program (including Contributions) may always be distributed subject to the version of the Agreement under which it was received. In addition, after a new version of the Agreement is published, Contributor may elect to distribute the Program (including its Contributions) under the new version. Except as expressly stated in Sections 2(a) and 2(b) above, Recipient receives no rights or licenses to the intellectual property of any Contributor under this Agreement, whether expressly, by implication, estoppel or otherwise. All rights in the Program not expressly granted under this Agreement are reserved.

This Agreement is governed by the laws of the State of New York and the intellectual property laws of the United States of America. No party to this Agreement will bring a legal action under this Agreement more than one year after the cause of action arose. Each party waives its rights to a jury trial in any resulting litigation.

Silk Icons

```
Creative Commons Attribution 2.5 License

THE WORK (AS DEFINED BELOW) IS PROVIDED UNDER THE TERMS OF THIS
CREATIVE COMMONS PUBLIC LICENSE ("CCPL" OR "LICENSE"). THE WORK IS
PROTECTED BY COPYRIGHT AND/OR OTHER APPLICABLE LAW. ANY USE OF THE
WORK OTHER THAN AS AUTHORIZED UNDER THIS LICENSE OR COPYRIGHT LAW IS
PROHIBITED.

BY EXERCISING ANY RIGHTS TO THE WORK PROVIDED HERE, YOU ACCEPT AND
AGREE TO BE BOUND BY THE TERMS OF THIS LICENSE. THE LICENSOR GRANTS
YOU THE RIGHTS CONTAINED HERE IN CONSIDERATION OF YOUR ACCEPTANCE OF
SUCH TERMS AND CONDITIONS.

1. Definitions

   1. "Collective Work" means a work, such as a periodical issue,
   anthology or encyclopedia, in which the Work in its entirety in
   unmodified form, along with a number of other contributions,
   constituting separate and independent works in themselves, are
   assembled into a collective whole. A work that constitutes a
   Collective Work will not be considered a Derivative Work (as
   defined below) for the purposes of this License.
```

2. "Derivative Work" means a work based upon the Work or upon the Work and other pre-existing works, such as a translation, musical arrangement, dramatization, fictionalization, motion picture version, sound recording, art reproduction, abridgment, condensation, or any other form in which the Work may be recast, transformed, or adapted, except that a work that constitutes a Collective Work will not be considered a Derivative Work for the purpose of this License. For the avoidance of doubt, where the Work is a musical composition or sound recording, the synchronization of the Work in timed-relation with a moving image ("synching") will be considered a Derivative Work for the purpose of this License.

3. "Licensor" means the individual or entity that offers the Work under the terms of this License.

4. "Original Author" means the individual or entity who created the Work.

5. "Work" means the copyrightable work of authorship offered under the terms of this License.

6. "You" means an individual or entity exercising rights under this License who has not previously violated the terms of this License with respect to the Work, or who has received express permission from the Licensor to exercise rights under this License despite a previous violation.

2. Fair Use Rights. Nothing in this license is intended to reduce, limit, or restrict any rights arising from fair use, first sale or other limitations on the exclusive rights of the copyright owner under copyright law or other applicable laws.

3. License Grant. Subject to the terms and conditions of this License, Licensor hereby grants You a worldwide, royalty-free, non-exclusive, perpetual (for the duration of the applicable copyright) license to exercise the rights in the Work as stated below:

   1. to reproduce the Work, to incorporate the Work into one or more Collective Works, and to reproduce the Work as incorporated in the Collective Works;

   2. to create and reproduce Derivative Works;

   3. to distribute copies or phonorecords of, display publicly, perform publicly, and perform publicly by means of a digital audio transmission the Work including as incorporated in Collective Works;

   4. to distribute copies or phonorecords of, display publicly, perform publicly, and perform publicly by means of a digital audio transmission Derivative Works.

   5. For the avoidance of doubt, where the work is a musical

composition:

i. Performance Royalties Under Blanket Licenses. Licensor waives the exclusive right to collect, whether individually or via a performance rights society (e.g. ASCAP, BMI, SESAC), royalties for the public performance or public digital performance (e.g. webcast) of the Work.

ii. Mechanical Rights and Statutory Royalties. Licensor waives the exclusive right to collect, whether individually or via a music rights agency or designated agent (e.g. Harry Fox Agency), royalties for any phonorecord You create from the Work ("cover version") and distribute, subject to the compulsory license created by 17 USC Section 115 of the US Copyright Act (or the equivalent in other jurisdictions).

6. Webcasting Rights and Statutory Royalties. For the avoidance of doubt, where the Work is a sound recording, Licensor waives the exclusive right to collect, whether individually or via a performance-rights society (e.g. SoundExchange), royalties for the public digital performance (e.g. webcast) of the Work, subject to the compulsory license created by 17 USC Section 114 of the US Copyright Act (or the equivalent in other jurisdictions).

The above rights may be exercised in all media and formats whether now known or hereafter devised. The above rights include the right to make such modifications as are technically necessary to exercise the rights in other media and formats. All rights not expressly granted by Licensor are hereby reserved.

4. Restrictions.The license granted in Section 3 above is expressly made subject to and limited by the following restrictions:

1. You may distribute, publicly display, publicly perform, or publicly digitally perform the Work only under the terms of this License, and You must include a copy of, or the Uniform Resource Identifier for, this License with every copy or phonorecord of the Work You distribute, publicly display, publicly perform, or publicly digitally perform. You may not offer or impose any terms on the Work that alter or restrict the terms of this License or the recipients' exercise of the rights granted hereunder. You may not sublicense the Work. You must keep intact all notices that refer to this License and to the disclaimer of warranties. You may not distribute, publicly display, publicly perform, or publicly digitally perform the Work with any technological measures that control access or use of the Work in a manner inconsistent with the terms of this License Agreement. The above applies to the Work as incorporated in a Collective Work, but this does not require the Collective Work apart from the Work itself to be made subject to the terms of this License. If You create a Collective Work, upon notice from any Licensor You must, to the extent practicable, remove from the Collective Work any credit as required by clause 4(b), as requested. If You create a Derivative Work, upon notice from any Licensor You must, to the extent practicable, remove from the Derivative Work any credit as required by clause 4(b), as

requested.

2. If you distribute, publicly display, publicly perform, or publicly digitally perform the Work or any Derivative Works or Collective Works, You must keep intact all copyright notices for the Work and provide, reasonable to the medium or means You are utilizing: (i) the name of the Original Author (or pseudonym, if applicable) if supplied, and/or (ii) if the Original Author and/or Licensor designate another party or parties (e.g. a sponsor institute, publishing entity, journal) for attribution in Licensor's copyright notice, terms of service or by other reasonable means, the name of such party or parties; the title of the Work if supplied; to the extent reasonably practicable, the Uniform Resource Identifier, if any, that Licensor specifies to be associated with the Work, unless such URI does not refer to the copyright notice or licensing information for the Work; and in the case of a Derivative Work, a credit identifying the use of the Work in the Derivative Work (e.g., "French translation of the Work by Original Author," or "Screenplay based on original Work by Original Author"). Such credit may be implemented in any reasonable manner; provided, however, that in the case of a Derivative Work or Collective Work, at a minimum such credit will appear where any other comparable authorship credit appears and in a manner at least as prominent as such other comparable authorship credit.

5. Representations, Warranties and Disclaimer

UNLESS OTHERWISE MUTUALLY AGREED TO BY THE PARTIES IN WRITING, LICENSOR OFFERS THE WORK AS-IS AND MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE WORK, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, WARRANTIES OF TITLE, MERCHANTIBILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, OR THE ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OF ABSENCE OF ERRORS, WHETHER OR NOT DISCOVERABLE. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO SUCH EXCLUSION MAY NOT APPLY TO YOU.

6. Limitation on Liability. EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, IN NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. Termination

1. This License and the rights granted hereunder will terminate automatically upon any breach by You of the terms of this License. Individuals or entities who have received Derivative Works or Collective Works from You under this License, however, will not have their licenses terminated provided such individuals or entities remain in full compliance with those licenses. Sections 1, 2, 5, 6, 7, and 8 will survive any termination of this License.

2. Subject to the above terms and conditions, the license granted

here is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, Licensor reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.

8. Miscellaneous

   1. Each time You distribute or publicly digitally perform the Work or a Collective Work, the Licensor offers to the recipient a license to the Work on the same terms and conditions as the license granted to You under this License.

   2. Each time You distribute or publicly digitally perform a Derivative Work, Licensor offers to the recipient a license to the original Work on the same terms and conditions as the license granted to You under this License.

   3. If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties to this agreement, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

   4. No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent.

   5. This License constitutes the entire agreement between the parties with respect to the Work licensed here. There are no understandings, agreements or representations with respect to the Work not specified here. Licensor shall not be bound by any additional provisions that may appear in any communication from You. This License may not be modified without the mutual written agreement of the Licensor and You.