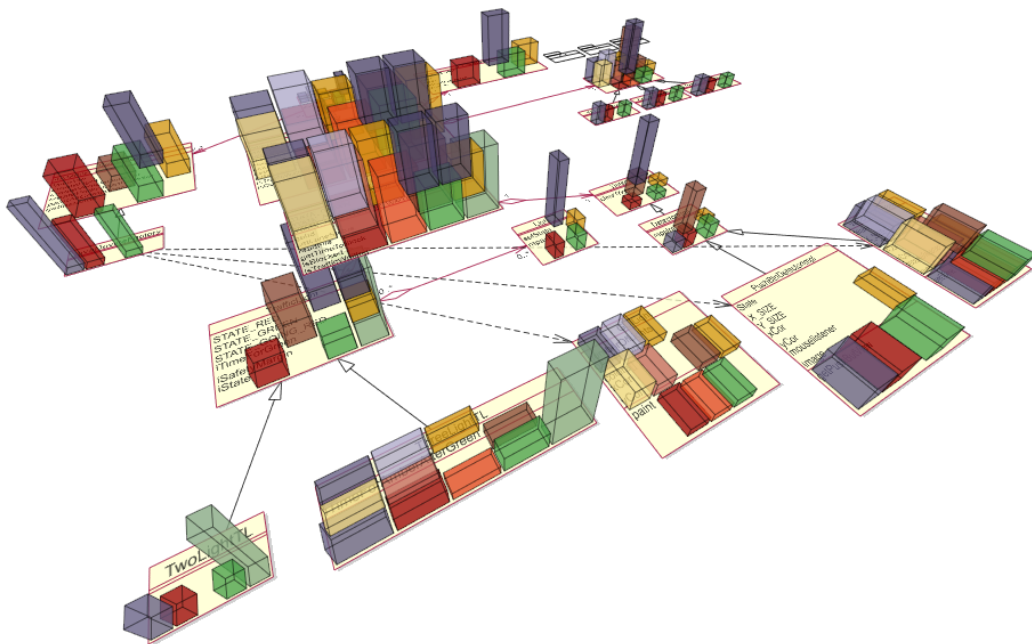


# MetricView

Visualization of Software Metrics on UML Architectures



Maurice Termeer

16th January 2005

# Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Getting Started</b>	<b>3</b>
2.1	Installation Procedure . . . . .	3
2.2	Running the Program . . . . .	3
<b>3</b>	<b>User Guide</b>	<b>4</b>
3.1	Introduction to Metrics . . . . .	4
3.2	The User Interface . . . . .	5
3.2.1	Toolbar Buttons . . . . .	6
3.2.2	Window Controls . . . . .	7
3.2.3	Window Interaction . . . . .	8
3.2.4	Metric Related Controls . . . . .	8
3.2.5	Global Configuration Options . . . . .	9
3.2.6	Metric Configuration Options . . . . .	11
3.3	Global Functionality . . . . .	11
3.3.1	Supported UML Diagram Types . . . . .	11
3.3.2	Supported File Formats . . . . .	11
3.3.3	Commandline Arguments . . . . .	12
3.4	Hardware Requirements . . . . .	13
<b>4</b>	<b>Developers Guide</b>	<b>14</b>
4.1	Global Architecture . . . . .	14
4.1.1	UML Metamodel Data-structure . . . . .	14
4.1.2	Metrics Data-structure Architecture . . . . .	14
4.1.3	Model Visualizers Architecture . . . . .	14
4.1.4	The XMI Parser and the UML Metamodel Data-structure . . . . .	16
4.1.5	Model Related Data Management . . . . .	16
4.2	Programming Environment . . . . .	17
4.2.1	Compiling the Code . . . . .	17
4.3	Possible Extensions and Improvements . . . . .	18
4.3.1	Placement of Configurations . . . . .	18
4.3.2	Integrate SAAT and MetricView . . . . .	18
4.3.3	Using a Layout Engine . . . . .	19
4.3.4	Abstract from UML . . . . .	19
4.3.5	Linking diagrams . . . . .	19
4.3.6	Visualizing other data . . . . .	19
	<b>Bibliography</b>	<b>20</b>



# Chapter 2

# Getting Started

## 2.1 Installation Procedure

The Microsoft Windows version of MetricView comes with an installation program. The installer was created with an open-source program called NSIS, the Nullsoft Scriptable Install System, and provided a common installer user interface. Following the direction should give no problems. An uninstaller is also supplied, so removing the program is just as easy.



## 2.2 Running the Program

Once the program is installed, it can be run by for example using the icons placed in the start menu. The user is presented with the main user interface. An explanation of this interface can be found in 3.2. To load a UML model and display some metrics on them, perform the following steps.

1. Use File → Openmodel... from the main menu to load a UML model from an XMI file.
2. Use File → Openmetrics... to load a corresponding metrics file, as produced by the SAAT. If the metrics file only differs from the XMI file in its extension, this step isn't necessary since the metrics file is loaded automatically in that case.
3. Select a diagram from the model view window (top-left treeview).
4. Hover over the elements to see which metrics are defined on those elements.
5. Check some of the metrics in the metrics listbox (bottom-left) to display them over the UML model.

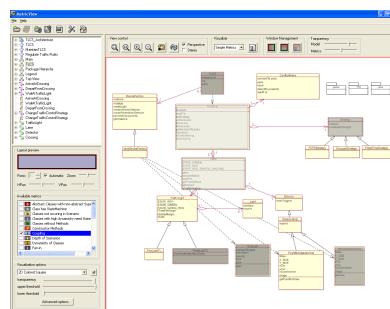


Figure 2.1: The main user interface with a UML model and metrics loaded.

# Chapter 3

## User Guide

The user guide is aimed at the end users of MetricView, those who will use the program to map metrics onto their UML diagrams. Developers who want to extend functionality should have a look at the developers guide instead. The user guide give a short introduction into metrics, an explanation of the user interface and an overview of the available features.

### 3.1 Introduction to Metrics

It is assumed that the reader is familiar with both UML models and diagrams as well as software metrics, as this is required to interpret the visualizations produced MetricView. The UML used in is quite minimal. It basically only consists of displaying a few types of diagrams.

Since the whole world around metrics is not yet standardized, here is a short introduction into the metrics system used in MetricView. On the top level, there are two types of metrics distinguished, *global* metrics and *element metrics*. Global metrics say something of the model as a whole, such as the number of classes in the model. Element metrics are values associated with a set of elements of a UML model. Currently two types of element metrics are identified; boolean and integer metrics. An element metric is defined on a subset of the model elements of the UML model. For each model element it is defined on, there is a  $(name, value)$  pair. The name of the metric is unique for the whole model. Every instance of the same metric contains the same name; it acts as an identifier of the metric. This means that each model element has a list of  $(name, value)$  pairs containing the metrics and the corresponding values, but this list differs from element to element, as not all metrics are defined for all model elements.

The goal of MetricView is to visualize this metric data (specifically the element metrics) onto the original UML diagrams to allow an easy, interactive exploration of metrics on UML diagrams.

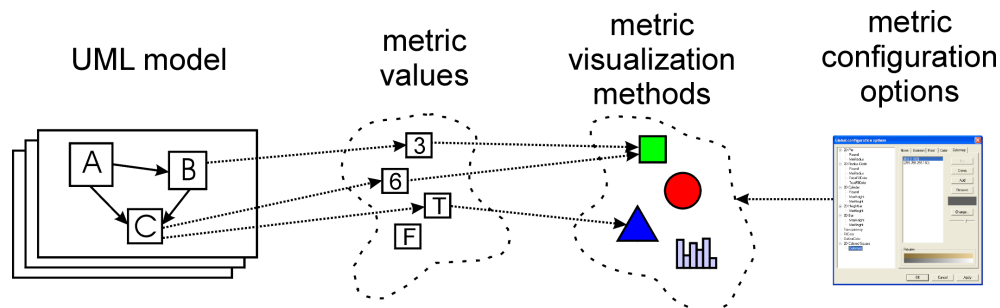


Figure 3.1: An overview of how various elements in MetricView are linked together to form a visualization.

One of the ways this interaction is achieved is by offering a variety of visualization methods. Any combination of metrics can be simultaneously visualized onto a diagram and each metric can have its own

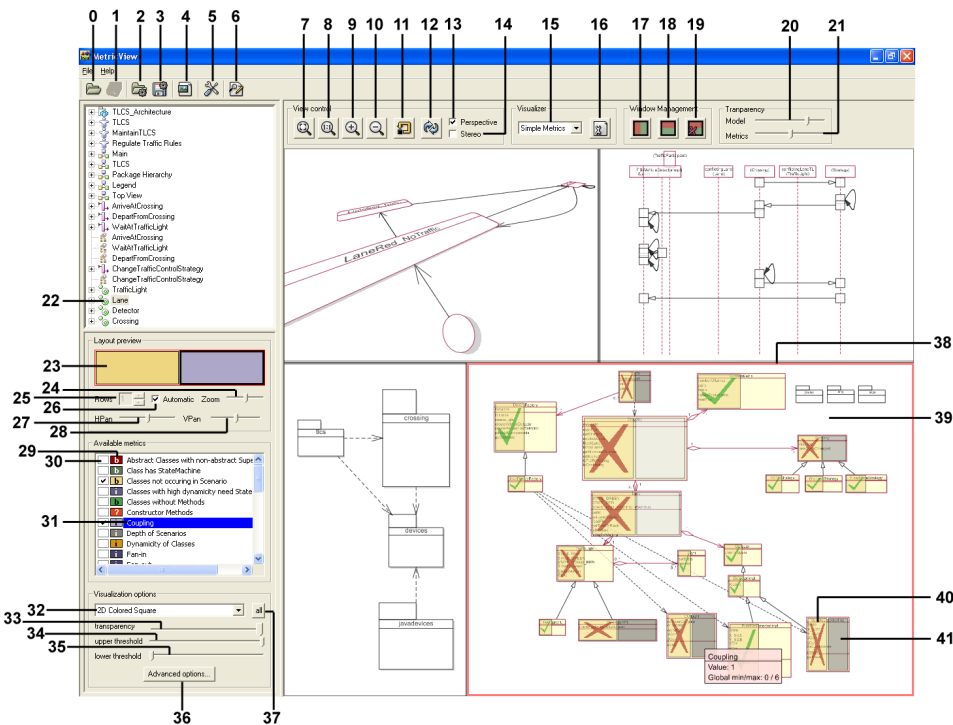


Figure 3.2: The main user interface of MetricView.

visualization method. Of course each metric also has its own visualization options, also including settings for each of the possible visualization methods. An overview of the mappings between UML elements, metric values, visualization methods and configuration settings is given in figure 3.1. Note that not all mappings are possible. For example, some visualization methods are restricted to element metrics of certain type. As shown in figure 3.1, a single model element may have multiple element metrics and a visualization method may be used by multiple metrics. Each metric may only have one visualization method though. What is not entirely clear is that the configuration options are defined per-metric. How all these mappings can be altered is described in the next section, which gives detailed description of the user interface.

## 3.2 The User Interface

Besides a few dialog boxes, all user interaction is performed from the main user interface. This interface, with a UML model and corresponding metrics loaded and displayed, is shown in figure 3.2. Below is a table that explains the numbers in the figure. Some of the options are described in more detail later on.

First a global description of the main user interface is given. The big area that covers the lower right part of the screen is called the *canvas* (22) and is the place where all drawing takes place. It is subdivided into one or more *windows*, separated by the window separators (35). Each window can display a diagram, which can be selected from the *model explorer* (23), the listbox on the upper left part of the screen. On top of the canvas are controls to interact with the windows and canvas (7 to 21). Finally, on the lower left part of the screen are controls to deal with visualizing metrics (24 to 34). Each part will be described in more detail, highlighting useful features on the way. The table below gives a short explanation for each control and optionally a subsection in which it is explained in more detail.

Toolbar buttons	(3.2.1)
(0)	Load a UML model from an XMI file.
(1)	Load metrics from a metrics file.

- (2) Load configuration settings from a file.
- (3) Save the current configuration settings to a file.
- (4) Save the current contents of the canvas as an image file.
- (5) Modify global configuration settings.
- (6) Give an overview of the global metrics currently loaded.

---

Window related controls (3.2.2)

- (7) Change zoom level such that the current diagram just fits in the window.
- (8) Set the zoom level of the active window to unit level.
- (9) Zoom in on currently active window.
- (10) Zoom out of currently active window.
- (11) Reset zoom level, rotation and pan of currently active window.
- (12) Refresh display of all windows.
- (13) Toggle between perspective projection and orthogonal projection.
- (14) Toggle stereo rendering mode.
- (15) The UML model and metrics layout visualizer for the current window.
- (16) Modify global visualizer options.
- (17) Split window vertically.
- (18) Split window horizontally.
- (19) Delete window (only possible if there is more than one window).
- (20) Control UML model transparency level.
- (21) Control metrics transparency level.
- (22) UML model explorer and diagram listbox.

---

Metrics related controls (3.2.6)

- (23) Active metrics topology overview.
- (24) Metric layout zoom level.
- (25) Number of columns in metrics layout.
- (26) Toggle automatic computation of columns in metrics layout.
- (27) Metric layout horizontal panning level.
- (28) Metric layout vertical panning level.
- (29) Metric type icon in metrics listbox.
- (30) Toggle metric to be active.
- (31) Selected metric.
- (32) Selected metric visualization method.
- (33) Selected metric transparency level.
- (34) Upper threshold for active metric.
- (35) Lower threshold for active metric.
- (36) Configure advanced options for selected metric.
- (37) Set current visualization method for all active metrics of same type.

---

Windowing system (3.2.2)

- (38) A window separator border.
  - (39) The window drawing area.
  - (40) An example of a boolean metric.
  - (41) An example of an integer metric.
- 

### 3.2.1 Toolbar Buttons

Most of the options from the toolbar can also be accessed from the pull-down menu, but using the buttons is usually faster. The left-most button (0) opens a UML model from an XMI file. For an overview of the supported file formats for UML models and metrics, see section 3.3.2. The next button (1) is only available when a UML model is loaded and loads a metrics file. Then there are two buttons (2 and 3) that load respectively save the global configuration settings (which are model-independent) from or to a file. Next to those is the button (4) that exports the current view to an image file. When exporting an image, the contents entire canvas (all the windows, including the separators) is exported as an image. The

currently supported image file formats are PNG and JPEG. There are no configuration options for any of the file formats. To ensure lossless image compression, choose the PNG image file format. Then there is a button (5) to access the global configuration options. For an overview of the dialog for global configuration settings, see section 3.2.5. Finally, the last button (6) gives a short overview of the global metrics present in the currently loaded metrics.

### 3.2.2 Window Controls

To be able to display multiple diagrams simultaneously, a windowing system is implemented. The canvas (the entire drawing area), can be split into a (virtually infinite) number of windows. Figure 3.3 gives an example of this functionality in action. There always is exactly one active window. This window can be recognized by a (by default) red border (the border color and size can be modified in the global configuration). To change the active window, simply click somewhere on a not active window to make it the active window.

On top of the canvas these is a collection of controls, organized in categories. The first category, view control, contains controls that only affect the active window. Starting on the left, the first four buttons (7 to 10) control the zoom level of the window, each represented by some sort of magnifying glass. Next there is a button that resets orientation of the window (11). Not only the zoom level is restored, but also the rotation and panning. Next there is a button to refresh the display of the window (12), in case something got messed up.

Then there are two checkboxes, which are related to the 3D display capabilities. The upper checkbox (13) toggles between perspective and orthogonal projection. The benefits of both modes will become clear when viewing the model and possibly metrics from an angle. With perspective projection, depth is very natural, but objects in the distance appear smaller. When using orthogonal project, the depth perception is distorted, but objects are not scaled down as they move into the distance. Also, when metrics are displayed using a 3-dimensional visualization method, the top-view looks completely 2-dimensional. For more information on how to control the viewing angle and panning of a window, see section 3.2.3.

The lower checkbox (14) toggles stereo viewing mode. When in stereo mode, the image is displayed as an anaglyph. This means that the entire scene is drawn twice, once in red, viewed a little bit from the right, and once in cyan, viewed a little bit from the left. If the user wears suitable 3D-glasses (a red filter over the left eye and a cyan filter over the right eye), depth perception should be more realistic.

The next category of controls is the visualization method. The actual visualization method itself also applies to the active window only. It controls the way the UML model is displayed and how the metrics are mapped onto the UML model. Next to the combobox (15) is a button (16) to access some global options of the visualizers. The functionality of this button is the same as the global configuration options button and is thus not specific to the active window.

Then there is the category for window splitting and deletion. The left-most two buttons control window splitting. When pressed, the active window is split into two windows, either vertically (17, left button) or horizontally (18, right button). After the split, there is a separator between the two windows. This separator is initially exactly in the middle of the original window, but can be dragged around to resize the windows. The third button (19) deletes the active window and can thus only be pressed if there are at least two windows. When a window is deleted, a neighboring window(-tree) takes up the space of the deleted window and (a child of that tree) becomes the new active window.

Finally there are two sliders to control the transparency level of the UML model and the metrics. The upper slider (20) controls the global transparency of the UML model, while the lower slider (21) controls the global transparency of all metrics. The lower slider is particularly useful for showing metrics outside certain threshold values. See section 3.2.4 for more details on metric threshold values.

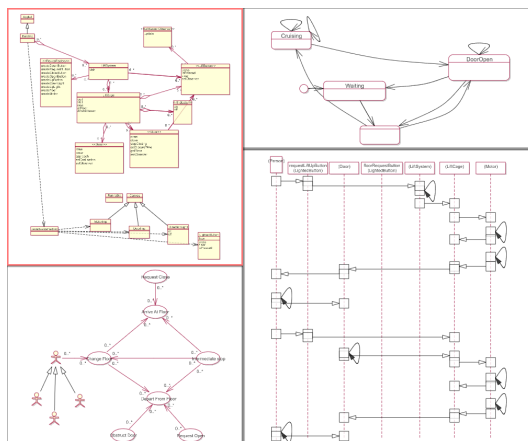


Figure 3.3: The windowing system in action.

### 3.2.3 Window Interaction

#### Resizing Windows

Besides the buttons, there are other ways to interact with a window. One of such actions is resizing a window. When the canvas is split into multiple windows, there is a window separator (38) between each pair of windows. By default, this separator is in the middle. To resize the two windows, simply drag the window separator around by moving the mouse over the separator (notice that the cursor changes into a resize cursor), press and hold the left mouse button (in case of a right-hand mouse configuration, as will be the assumption for the rest of the document) and drag the mouse around.

#### Changing a Windows Orientation

Besides the zoom level, the orientation of the “camera” of a window consists also of panning and rotation. The zoom level can be controlled with some buttons described earlier, but also with the scroll-wheel of the mouse. This is usually more convenient. When pointing somewhere inside the active window and dragging the mouse around while the middle mouse button (usually the scroll-wheel) is pressed adjusts the panning. Dragging the mouse around while keeping the middle mouse button and the shift key on the keyboard pressed adjusts the rotation. Initially, the center of the screen is the origin and the camera looks in the positive  $z$ -direction, so the model is displayed in the  $(x, y)$ -plane. While changing rotation, horizontal movement controls rotation around the  $z$ -axis, while vertical movement controls rotation around the  $x$ -axis. This allows for quick and easy modification of the orientation of a window.

#### Hovering over Elements

When no buttons are pressed, moving the mouse over any of the element draw inside the window causes a pop-up to appear near the mouse position. It contains information about the element. In case of a model element, it displays the name of the element followed by a list of metrics that are available for that element. In the case of a metric, it displays the name and value of the metric. If the metric is of an integer type, the global minimum and maximum are also displayed.

#### Selecting Elements

A selection mechanism is also available. Note that only model elements can be selected. To select a model element, simply click on it. A blue overlay will be painted over the element. To select multiple elements, hold shift while clicking on the model elements. Holding control while clicking several times on the same location allows a user to select an element that is behind another element. Besides a visual change, the selection is currently useless. There are no actions that can be performed on selected elements.

#### Modifying Elements Layout

When some elements are selected, they can be repositioned or resized. To change the position of all selected elements, keep the alt key on the keyboard and the left mouse button pressed while dragging the mouse around. To resize all selected elements, keep both the alt and control key on the keyboard as well as the left mouse button pressed while dragging the mouse around. Moving and resizing elements also works for some relations. Moving and resizing a transition in a state chart diagram adjusts the curvature of a relation for example, while in a message sequence diagram only the vertical position of a message can be altered.

### 3.2.4 Metric Related Controls

The lower left part of the screen is filled with several controls to control the metrics visualization. On top is a rectangle that gives an overview of the currently active metrics and their layout (23), labeled the *layout preview*. When one or more metrics are active, they are represented inside this rectangle by a smaller rectangle colored with the metrics own color. The layout corresponds to the layout of the metrics on the model elements. Clicking on a metric in this topology overview selects that metric in the metric listbox,

right-clicking on it deactivates the metric. Below the layout preview control are two widgets to control the layout schema (25 and 26). When the automatic layout is used, The number of columns is determined based on the number of active metrics. Then there are three sliders to control the transformations of the mapping of the metrics to the elements. The zoom (24), horizontal panning (27) and vertical panning (28) can be controlled.

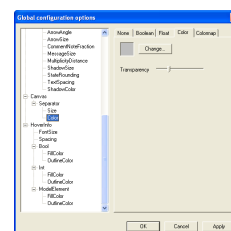
Below is the metrics listbox. It contains a list of currently loaded metrics. In front of each metric is a checkbox (30), to indicate whether this metric is active or not. Clicking on it will toggle the metrics active status. Next is a small square in the color of the metric (29) that will be used to draw the metric (this color is also used in the layout preview). Inside that same square also is a symbol which indicates the type of the metric. A “b” represents a boolean metric, an “i” represents a metric of an integer type and a “?” means the metric type is unknown. Finally there is the name of the metric.

Clicking on a metric causes it to become selected (31), the background of the metric name will turn blue. Below the listbox are several controls that deal with the selected metric. First there is the visualization method used to draw the metric on the model elements (32). As explained before, MetricView offers a palette of visualization methods to choose from in order to visualize a given metric. By selecting the desired method from the combobox, it becomes the active visualization method for the selected metric only. Then there are three sliders, the first to control the transparency level of the selected metric only (33), and two threshold sliders, for an upper (34) and a lower (35) threshold value. These last two sliders are only available for integer metrics. These threshold sliders are useful for quickly displaying high or low values of a metric. When the value of an integer metric is below the lower threshold or above the upper threshold, the global transparency setting is ignored. This means that if the global transparency level is set to a low value, the metrics outside the threshold range light up brighter than the other ones.

At the bottom there is an advanced options button (36), which allows to configure all the properties of a metric, such as color, but also all the configuration options for the different visualization options. For a more detailed description of these options, see section 3.2.6. Finally there is a small button with the caption “all” next to the visualization method combobox (37). Clicking this button causes the visualization method for all active metrics that are of the same type as the selected metric to be set to the visualization method of the selected metric. This is useful for changing the visualization method of all active metrics at the same time.

### 3.2.5 Global Configuration Options

There are quite a few global configuration options. Usually, these are not really important for the user to edit, so they are only accessible through a special dialog box. A snapshot of this dialogbox is given in the figure on the right. The left part of the dialog box contains a treview that has the options listed. Only the leaf nodes represent options. When clicking on an options, the right part of the dialog box changes to an interface to modify the value of the selected option. Below is a list of the available options with a description for each option.



#### Visualizer

##### General

Abstractions	Enable / disable drawing of abstractions.
Associations	Enable / disable drawing of associations.
Attributes	Enable / disable drawing of attributes in classifiers.
Comments	Enable / disable drawing of comments.
Dependencies	Enable / disable drawing of dependencies.
ElementNames	Enable / disable drawing of element names.
Generalizations	Enable / disable drawing of generalizations.

##### Legend

Enable	Enable / disable drawing of the legend.
FontSize	The fontsize used in the legend.
Spacing	The spacing in pixels between the elements in the legend.
Transparency	The transparency level of the legend.

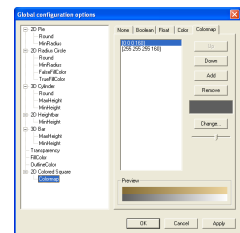
FillColor	The background color of the legend.
FontColor	The color of the text in the legend.
OutlineColor	The outline color of the legend.
Messages	Enable / disable drawing of messages.
RelationNames	Enable / disable drawing of relation names.
Shadows	Enable / disable drawing of shadows of elements.
Transitions	Enable / disable drawing of transitions.
RelationWidth	Modifies the virtual width of relations, used when selecting relations.
Stereo	
EyeSeparation	The distance between the two virtual eyes, when in stereo mode.
FocusDistance	The focus distance, the distance around which the view will be rotated in stereo mode.
SelectColor	The color of the overlay drawn over model elements when they are selected.
SimpleMetrics	
Spacing	The spacing between the border of the model elements and the metrics visualization.
UMLModel	
ArrowAngle	The angle between the lines of the arrowheads in certain types of relations.
ArrowSize	The length of the arrowhead in certain types of relations.
CommentNoteFraction	The fraction of the size of a comment used for the folded corner.
MessageSize	The height of a message begin and endpoint.
MultiplicityDistance	The offset distance where to draw the multiplicity near an association.
ShadowSize	The size of the shadow for elements, in pixels.
StateRounding	The fraction of the size of a state that is used for the radius of the rounded corners.
TextSpacing	The space between text and other graphics objects.
ShadowColor	The color of the shadow of elements.
Canvas	
Separator	
Size	Width of the window separator lines, in pixels.
Color	Color of the window separator lines.
HoverInfo	
FontSize	Fontsize used in hoverinfo boxes.
Spacing	Textspacing used in hoverinfo boxes.
Bool	
FillColor	Fill color used for hoverinfo boxes of bool metrics.
OutlineColor	Outline color used for hoverinfo boxes of bool metrics.
Int	
FillColor	Fill color used for hoverinfo boxes of int metrics.
OutlineColor	Outline color used for hoverinfo boxes of int metrics.
ModelElement	
FillColor	Fill color used for hoverinfo boxes of model elements.
OutlineColor	Outline color used for hoverinfo boxes of model elements.

### Automatic configuration loading

On the startup of MetricView, existence of a file “default.cfg” in the program directory is checked. If found, this file is loaded at startup to act as a default configuration. This is useful for modifying the default values of the global configuration options at startup.

### 3.2.6 Metric Configuration Options

Each metric has a set of options, such as color, but also options that the different visualizers use. One could think of the value used for checkmarks of boolean metrics. All these options are controlled from a dialog under the advanced options button, which gives a similar dialog as the global configuration dialog. Since the visualizers available depend on the type of the metric, so do the options. Most options are quite trivial, such as the color or such. The dialog used is in fact similar to that of the global configuration options, only different options are presented. Since the options are all quite trivial, no further explanation is given.



## 3.3 Global Functionality

Besides a user interface, MetricView also offers some functionality that can be controlled using the user interface. This section gives an overview of some of this functionality and highlights interesting details.

### 3.3.1 Supported UML Diagram Types

Since the entire UML specification is quite complex, not all elements and diagram types will be supported. The focus is on support for the following diagram types.

- Class diagrams
- Sequence diagrams
- Use Case diagrams
- State diagrams
- Collaboration diagrams

Again due to some time limits, not all information according to the UML specification is available in each diagram. Only the necessary elements will be extracted and displayed. For example, a class will have only a name and a list of methods and attributes. These methods and attributes have a visibility property, types and optionally parameters. Other properties such as tagged values, template parameters, participant information et cetera is not displayed (or even extracted from the input document).

### 3.3.2 Supported File Formats

There are three types of files that the tool can interact with: XMI files containing a UML model, metrics files and its own configuration files. The latter is not a public file format and thus needs no description. Although it is a text-based file (and thus somewhat human-readable), it is not in XML format and modifying the file with a text editor is discouraged. The configuration settings include the global configuration settings, the window layout and setup, including orientation and the diagrams loaded in them, all options for the metrics and the list of active metrics. The other two file formats are dealt with by other applications as well, so some additional information is provided.

### UML Input Format Specification

The UML models are imported from XML Metadata Interchange (XMI) files, following the official XMI specification version 1.2 of the Object Management Group, as mentioned in [3]. Along with this, the UML version 1.3 metamodel is implemented. A DTD for this metamodel is also provided by OMG, namely [2]. Since the entire XMI specification and UML metamodel are both quite complex, they are not fully implemented. Instead, the most common elements are extracted from the documents. This gives enough information to visualize the basic UML models and project the metrics on them.

Although these standards are already somewhat aged, they are still more common than their successors, XMI 2.0 and UML 2.0. At the time of writing, the UML 2.0 standard is still not yet released as final, but more importantly, there is only a very limited tool set that supports these file formats. So for compatibilities sake, import filters for the old file formats have been implemented.

Moving towards XMI 2.0 should not be a major problem, as very few XMI related elements are implemented and the new XMI format is not that different from its predecessor, looking at the parts used here. Moving to UML 2.0 will be a considerable amount of work. The UML 2.0 metamodel follows the same structure, so parsing the model will be like extending the parser. The diagram information however, is stored in a completely new and different way that in no way resembles the current (very limited) way. Much more details are stored, which allow for much more complicated diagrams and positioning of elements (especially relations between elements such as dependencies and associations).

### Metric Input Format Specification

The metric data is be read from simple plain text files. There is no known official file format specification (or even file extension). The metrics import filter was built looking at example input files. As the tools that generate the metrics are developed by SAN itself, any issues are easily resolved. In fact, the file format changed several times during the project to make certain extensions possible.

Three types of metrics are distinguished. These metrics are identified by means of matching them to a regular expression. A *global metric* is a single number. An example is the number of classes in the model. Next there are metrics defined for a number of model elements, usually of the same type. Currently two *element metric* types are defined, *integer metrics* and *boolean metrics*. Boolean metrics consists of a list of elements that have a certain property. For example, the classes that do not have any subclasses in the model. Integer metrics are a mapping of a model element to a number. One could think of the number of methods of each class for instance.

The metric type is automatically detected and with each metric type, no distinction is made. This means that within one type of metric, all metrics are treated equally. Since there is no metric-specific code, new metrics can be added without modifying the visualization tool. This also holds for the visualization methods. All metric visualization methods are only bound to a certain metric type. So any new metrics can immediately visualized with existing methods.

The metrics file is not checked to be corresponding to the currently used model. Any metrics defined on elements not present in the current model are discarded.

### Automatic Metrics Loading

It is common to have metrics files with the same name as the corresponding UML model XMI files, only with a different extension. When loading a model file, MetricView automatically checks for the existence of a metrics file with the same name as the XMI file, only with the “.txt” extension instead of “.xml” or “.xmi”. When found, this metrics file is also loaded. This makes loading data faster and easier.

### 3.3.3 Command-line Arguments

In order to instantiate MetricView from another application, MetricView supports a few command line arguments. Currently three options are recognized. All options have a short and a long version. The short options should be prefixed with a '/' on the Microsoft Windows platform and with a '-' in Unix like platforms. Long options should be prefixed with '--' (double dash) on all platforms. All options accept a string as an argument. For the short options there may optionally be a space between the option and the argument. For the long options, there should be an '=' between the options and the argument. Below is an overview of the accepted command line arguments.

Short option	Long option	Description
u	uml-model	Load UML model on startup
m	metrics	Load metrics on startup (requires a UML model)
c	config	Load configuration file on startup (overrides default.cfg)

## 3.4 Hardware Requirements

The hardware requirements are not really strict. It is vital that the machine provides good OpenGL support, but the graphics device need not be extremely fast nor be able to deal with the more advanced features of OpenGL (such as shaders), as the visualizations usually are not that complex from a graphics device point of view.

The UML models can become quite big and to be able to interact with this data a processor with equivalent speed of an Intel Pentium 4 running above 1.8GHz is recommended. MetricView was successfully used to visualize metrics onto UML models consisting of hundreds of classes, with XMI files being larger than 30 megabyte.

# Chapter 4

## Developers Guide

For those who want to extend the system, having a look at the architecture or knowing how to compile the stuff might be useful. Some knowledge of developing software is assumed.

### 4.1 Global Architecture

This section doesn't provide a complete architecture of every class in the system, as many classes are trivial or not important. Only the important parts and main idea behind the architecture is covered.

#### 4.1.1 UML Metamodel Data-structure

The left part of figure 4.1 shows the top layer of the UML metamodel data structure. The classes shown are only there for architectural purposes. The actual classes used are not in the diagram, as there are many and the structure is also mentioned in [4]. The main difference with the model in the official specification is that the `UMLElement` class contains an `id` field, which contains the XMI identifier of the element. Note that not only model elements have an XMI identifier, but also diagrams and diagram elements (so also elements that are not part of the model).

#### 4.1.2 Metrics Data-structure Architecture

The metrics have a significantly smaller data-structure, which is shown in the middle part of figure 4.1. The first categorization of metrics is the distinction between global and element metrics. Global metrics are metrics that say something about the whole system, while element metrics have a value for a set of model elements. The latter type is stored inside the UML metamodel structure, as `userdata` in the `userData` variable of the `UMLModelElement` class.

#### 4.1.3 Model Visualizers Architecture

To handle the drawing of the UML model and performing the layout of the metrics, there are the visualizer classes. These are shown in the right of figure 4.1. Since drawing a UML model is basically drawing a set of model elements and each model element should be drawn in a different way, deriving such a UML model visualizer of the `UMLModelVisitor` class is a logical thing to do. On the top is the `UMLModelVisualizer`, which does not contain any element-specific drawing code, but it does call the appropriate visitor methods when drawing a UML diagram. All elements are drawn using squares indicating the bounding boxes of the elements. The actual drawing code for the individual UML model elements is in `UMLModelVisualizerStandard`, which performs standard drawing of UML elements, that is, somewhat similar to other editors.

To map the metrics onto the UML model, there is the `MetricsVisualizer`. All this class does, is map the metrics onto the model elements and use a metrics `MetricVisualizer` (don't mix this up with `MetricsVisualizer`) to draw the metrics over the element. The actual drawing of metrics is done by a set

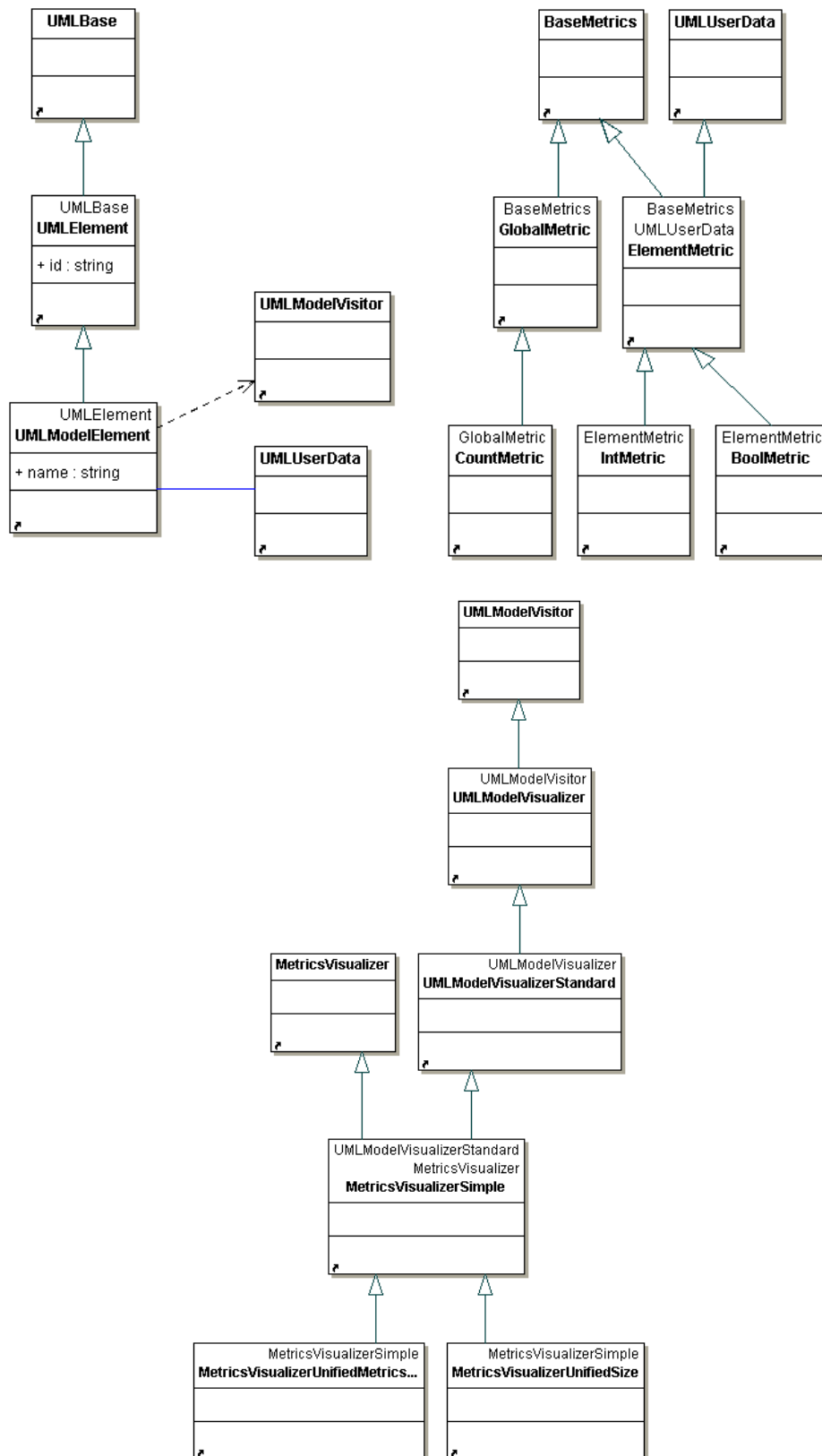


Figure 4.1: An overview of the architecture of the UML metamodel (left), the metrics data-structure (middle) and the UML model visualizers (right).

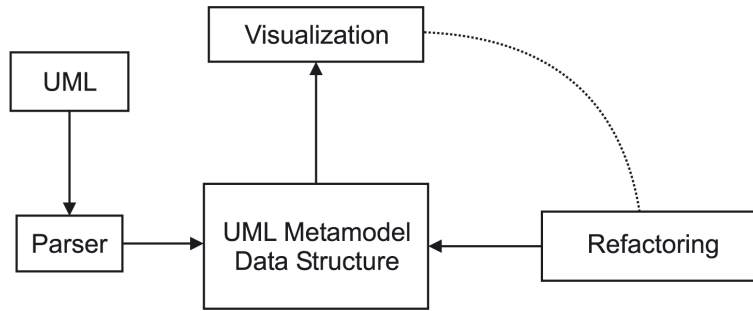


Figure 4.2: Global architecture with relation to the parser, data-structure and other components

of classes that all are a derivative of `MetricVisualizer`. The information supplied is the bounding box and the configuration data of the metric and the metric visualizer itself. Adding a new metric visualizer is not much work. The current ones are only about 20 lines per metric visualizer.

There are also two other visualizers, one that renders the model elements in a uniform size and one that renders only the metrics in a uniform size. Because there is no layout engine, the minimum element size in the diagram is taken at the uniform size, since that guarantees no overlaps.

#### 4.1.4 The XMI Parser and the UML Metamodel Data-structure

One big part of the project is writing an XMI parser. To extend the usability of this parser, it is written as an independent component that can communicate with the rest of the tool. Figure 4.2 gives an overview of the global architecture. The UML model can be imported in the tool by converting it to an XMI file, which can be parsed by the parser which is part of this project. The parser stores the UML model in a data structure that is based on the UML metamodel. This data structure is in essence a large collection of C++ classes. The visualization component uses this data-structure as an independent component. This means that all visualization related data concerning the model is not stored in the global data structure.

The main advantage of this approach is that possible other components such as a refactoring tool can easily make use of the same XMI parser and data structure. By even creating an interface between such a tool and the visualization component, the components may be integrated into one large system.

#### 4.1.5 Model Related Data Management

There are various kinds of data related to the UML model. Different components produce, modify or use different parts of this data. An overview is given in 4.3. For example, the XMI parser fill the data-structures of the model and the diagrams, while the visualizer uses all of the four data blocks. Because this data is used by so many different components, the data can be considered global. This calls for some data management module. However, the set of available data blocks is quite case-specific. Therefore, separate management modules for the different blocks of data are made.

The XMI parser makes use of a model manager, one that houses the UML model and the corresponding diagrams. The metrics parser makes use of a metrics manager, a manager that manages the metrics data. The visualizer uses, besides the model and metric manager, a selection manager, which, as one might guess, manages the currently selected model elements.

The data management modules as depicted in figure 4.3 are not big components, usually they just house a few variables. Also, an instance is copied to various places for easy and fast data access. This probably isn't the best solution (as it deviates quite some bit from the idea behind the figure), but it works nicely.

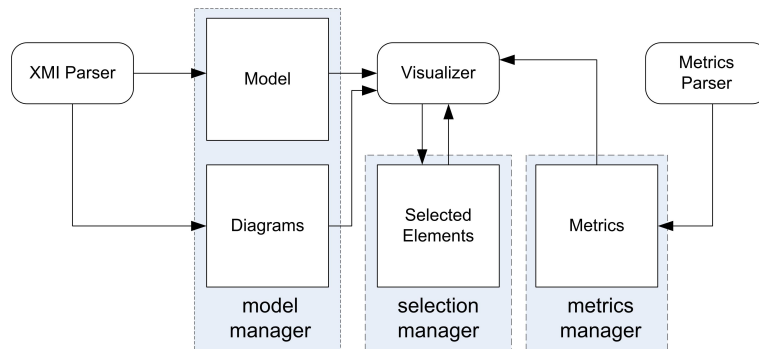


Figure 4.3: Model related data relations to components

## 4.2 Programming Environment

The entire product is programmed in the C++ programming language. The API's used are the boost library for some elementary features like string manipulation, tokenizing and regular expressions, wxWidgets for the graphical user interface, OpenGL for the more complex graphical operations (the main visualization tasks), Xerces-C++ as an XML parser for parsing the XMI files, FreeType for high quality text rendering and FTGL as an interface between FreeType and OpenGL. All of these libraries are more or less platform independent and thus the resulting product runs under both Microsoft Windows and Linux, and possibly others. The operating system and accompanied drivers should provide decent OpenGL support though.

Most of the work was done using the Microsoft Visual Studio C++ 2005 Express Beta suite. As the name says, it still is a beta edition, but it was available for free from the Microsoft website during development. Another benefit is that the code it generates is very efficient.

### 4.2.1 Compiling the Code

Compiling the code under Microsoft Windows is not as easy as it should be. First, a copy of every library used should be obtained. The current URLs of the libraries are given in table 4.1. When using the Microsoft Visual C++ 2005 Express Beta compiler, most projects contain solutions that can be opened to almost compile the project. Special care should be taken when choosing a build configuration. All libraries should be compiled using the **Multi-threaded DLL** runtime model. Note that when compiling wxWidgets, support for *wxGLCanvas* should be enabled (this has to be done manually) and when compiling something that uses this library, `WXUSINGDLL` should be defined. When using FTGL, `FTGL_LIBRARY_DYNAMIC` should be defined.

Library	Location
wxWidgets	<a href="http://www.wxwidgets.org">www.wxwidgets.org</a>
freetype	<a href="http://www.freetype.org">www.freetype.org</a>
FTGL	<a href="http://homepages.paradise.net.nz/henryj/code/">homepages.paradise.net.nz/henryj/code/</a>
Xerces-C++	<a href="http://xml.apache.org/xerces-c/">xml.apache.org/xerces-c/</a>
Boost	<a href="http://www.boost.org">www.boost.org</a>

Table 4.1: Some libraries used and their homepages.

When all the libraries are compiled, the solution of the MetricView project can be used to compile MetricView itself. The solution file is configured to compile all C++ files and link them against the library. Except for some preprocessor directives described above and the correct runtime model, there are no special options in the compile configuration.

When using the Microsoft Visual C++ 2005 (Express) Beta, be sure to update the compiler with the

Visual C++ 2005 Tools Refresh, as the first edition of the beta compiler contains some bugs which cause it to fail template instantiations using local type definitions.

### Compiling the Installer

When building the project is complete, an installer can be built. An installer-build script is made for the Nullsoft Scriptable Install System (NSIS), available for free from [nsis.sourceforge.net](http://nsis.sourceforge.net). Simply place the compiled binary plus all DLLs, the font file and the DTD in one directory and compile the NSIS script. This produces a single binary that can easily be distributed over the Internet.

## 4.3 Possible Extensions and Improvements

As the available time was limited, far from all possibilities have been explored considering visualizing metrics. Also, some of the features might be implemented in a different, possibly better, way.

### 4.3.1 Placement of Configurations

There are many elements such as windows and metrics that have their own configuration options, and there are also global configuration options. The current mapping of configuration options to elements may not be optimal. For example, the metric layout settings are currently per visualizer, not per window. This is a bit counter-intuitive. Also the set of active metrics is global, while perhaps making this a per-window setting as well would be better, as different windows could then contain different sets of metrics.

### 4.3.2 Integrate SAAT and MetricView

Currently the metrics are computed by the SAAT and imported again by MetricView. Directly computing the metrics from the modeldata that MetricView would be easier from a users point of view. This would also make it easier to add new metrics, as these are SQL-queries. Another approach would be to compute the metrics directly from the data-structures. This would remove the computational overhead that the SQL approach causes. The following is an example piece of code that compute a not so interesting integer metric for all the .. in the model. It needs access to the model manager and the metrics manager. It could be instantiated from the main class, that currently uses these components for loading models and metrics.

```
void ComputeMetrics(UMLModelManager *model, MetricsManager *metrics) {
    string name = "My Example Metric";

    metrics->MetricProperties[name] = new MetricProperties();

    int imin = 0, imax = 63;

    for (UMLResolveMap::iterator i=model->ResolveMap.begin(); i!=model->ResolveMap.end(); i++) {
        if (dynamic_cast<UMLClassifier *>((*i).second) != NULL) {
            IntMetric *metric = new IntMetric();
            metric->Name = name;
            metric->v = rand() % 64;
            metric->max = imax;
            metric->min = imin;

            dynamic_cast<UMLModelElement *>((*i).second)->userData.push_back(metric);
            metrics->ElementMetrics.push_back(metric);

            metrics->IntMinMax[name] = pair<int,int>(imin, imax);
        }
    }
}
```

}

### 4.3.3 Using a Layout Engine

One of the major restrictions of the possible visualization methods is caused by the lack of an automatic layout engine. Currently, only the layout inside the XMI could be used. This isn't that bad, as this design is often made by humans and if this information is used, the architect recognizes his or her work, making it easier to interpret the data. However, experimenting with different element sizes or abstracting from the size and shape of the standard UML elements (in order to visualize more data at once), is much easier and gives much better result if an automatic layout engine is available. For solving the 2D layout problem, GraphViz might be a good free tool, while GoVisual is a good option as a commercial library.

When using a 2D automatic layout engine, there is no dependency on the (often inconsistent) layout inside the XMI anymore. One could also think of compromises like using the bounding-box layout of the XMI for elements only, and let the layout engine solve the relations and such, while making own decisions on colors of elements. Letting the user move elements to restructure the diagram could also be a nice feature.

A whole different type of visualizations is possible using 3D layouts. Instead of visualizing the model inside the plane, using a 3D layout algorithm (such as force-directed layout) enables much more interesting visualizations. Some interesting options are presented in [1].

### 4.3.4 Abstract from UML

Using 3D layouts links to the next possible extension, to include more abstract visualizations of the UML model. Instead of using the standard representations, one could make up some weird looking glyphs, into which several metrics could be visualized at once. The downside is of course that the architect does not recognize his design anymore, but metrics may be mapped more clearly.

Another idea is to use the topology of a diagram. In case the architect globally knows which classes live where, one could use for instance a heightplot to visualize metrics over the UML diagram, making the diagram invisible, or partially hidden by means of transparency. Visualizations such as these are currently not possible, since these require a global visualization method instead of a per metric method.

### 4.3.5 Linking diagrams

Currently there is no connection between different diagrams, while the diagrams definitely have a connection in the model. For example, the classifier roles in a sequence diagram are linked to the classes of a class diagram. Somehow visualizing these links or mapping each others metrics on them are some useful extensions.

### 4.3.6 Visualizing other data

The tool now maps metrics onto UML diagrams, but one can think of other mappings as well. These are perhaps one of the larger extensions, but reverse-engineering code into a set of UML class diagrams and mapping the code-diagrams over the original diagrams and highlighting the differences could be a useful way to detect differences between the original design and the actual implementation.

# Bibliography

- [1] Tim Dwyer. Three dimensional uml using force directed layout. 2001.
- [2] Object Management Group. Xmi dtd 1.0 for uml 1.3, 2002.
- [3] Object Management Group. Xml metadata interchange (xmi) specification version 1.2. (formal/02-01-01), 2002.
- [4] Object Management Group. Unified modelling language specification. (formal/03-03-01), 2003. version 1.5.