Analysing models and model metrics

Learn how to identify model problems early and improve model governance

Steve Arnold

Large models, just like large bodies of code, deteriorate over time. The impact of making many changes to a model will, eventually, render it harder to change and understand. By regularly analysing model metrics, it is possible to maintain the model in a good state and ensure that the model has a consistent and well-formed structure. This article describes how to use the Model Metrics Analysis view in IBM® Rational® Software Modeler, Rational Software Architect, or Rational Systems Developer to analyse models and discover potential problem areas, such as classes with too many dependencies or dependents.

Model Metric Analysis Plug-in overview

The Model Metric Analysis Plug-in for IBM® Rational® software enables you to quickly get an overview of your models and to pinpoint areas of the model that may be developing problems. By quickly gathering metrics and displaying them in both a report view and distribution view, you can pinpoint problematic areas in the model in seconds. Figure 1 illustrates the clarity of the metrics information display.

Figure 1. Distribution and Details view

These views enable anyone to see potential problem areas of the system with a single glance. Staff members of each project or enterprise can define warning and error thresholds for each
metric by using the preferences screens, and these thresholds can then be shared amongst team members.

Model analysis will also generate Kiviat diagrams for most model elements, such as shown in Figure 2. This allows a modeler to see all of the gathered metrics for that element and decide whether it is just one metric that is problematic or whether that element is scoring badly in several ways.

**About Kiviat diagrams**

A Kiviat diagram takes several values or metrics for an element and displays them in a single diagram.

Each metric is marked on its own axis, and the points join together to form a shape. Where values exceed their tolerances or thresholds, the point on the axis is marked or coloured to highlight the discrepancy.

This form of diagram allows multiple values or metrics for one element to be viewed in one glance, providing a holistic view of the current state of that element. For instance, you might be interested in the number of attributes, number of operations, and number of dependencies for an element. Seeing all of that information on one diagram helps you make an informed decision as to whether there is a problem.

**Figure 2. Kiviat diagram**

The Model Metric Analysis Plug-in helps to automate the discovery of problems as much as possible. Most of the provided metrics hook into the IBM® Rational® Software Delivery Platform validation framework. This means that whenever a model is validated, the metrics will also be evaluated and a model warning will be generated for any element exceeding the current warning level for a specific metric. Finally, the Model Metric Analysis Plug-in can generate an offline HTML report which can be used in model reviews or for checks by senior architects to ensure that there are no problems with the current design.

**Plug-in installation**

The installation of the Model Metric Analysis Plug-in is straightforward.
1. Download the Reusable Asset Specification (RAS) file included with this article (see Downloads) to a location on your hard drive.
2. Start Rational Software Modeler (or IBM® Rational® Software Architect or IBM® Rational® Systems Developer), and select File > Import.
3. Choose RAS > RAS Asset, and click Next.
4. Browse to the downloaded RAS file and accept the warning about deployable plug-ins.
5. Click Next.
6. Select the feature, accept the license agreement, and then click Finish.
7. When prompted, restart to enable the plug-in.

Now switch to the Modeling perspective, and you should see the Model Metric Analysis view (Figure 3). If it is not visible, you can open it by selecting Window > Show View > Other and choosing Modeling > Model Metric Analysis from the list.

**Figure 3. Model Analysis view**

![Model Analysis view](image)

**Plug-in use**

You can interactively analyse your models by using the Model Analysis view.

**Interactive analysis**

The Model Analysis view enables you to generate metrics against each of your models and then view those metrics in either a detailed report or a distribution graph. The view also provides
Kiviat diagrams for reviewing multiple metrics for a particular element in the model. You can also generate a metric report that includes pages for all of the different metrics, as well as Kiviat diagrams for all appropriate elements.

Let's go through the steps required to analyse a model.

1. The first step is to generate some metrics.
   A. Make sure that the model that you want to analyse is open, and then select it in the dropdown box at the top of the Model Analysis view.
   B. Either select only the metrics that you wish to gather, or right-click on the metric list and choose **Select All** if you want to collect all of them.
   C. Press the gather metrics button or right-click and select **Gather Metrics**.

You should see updated values at the right of each metric, indicating both the total number of metrics and the average, as Figure 4 shows.

**Figure 4. Analysing a model**

2. Next, you need to analyse the metrics. You can choose from three views: details, a distribution graph, and a Kiviat diagram. The following subsections describe each of those, along with other options.

**Distribution Graph view**

When you select a single metric in the metric list, the distribution graph will show the number of elements that have a particular metric value. You can use this graph to quickly see if there is a problem with any metric. The bars are colour-coded to highlight the values that are at a warning or error level, as shown in Figure 5.
Figure 5. Distribution Graph view

Details view

Again, when you select a single metric in the metric list, the view will list all of the elements and their values for the selected metric. The list is colour-coded to highlight warnings and errors. You can sort it by clicking on the headings, and you can navigate from the list to the element in the project explorer by right-clicking and choosing Open in Model Explorer (see Figure 6).

Figure 6. Details view
Kiviat diagram view

The Kiviat diagram view shows many metrics for a particular element. It is a good way of understanding details of a particular element that you have concerns about.

1. First, in the metric list, you select the metrics that you want to see (you must select at least three).
2. Then, in the Kiviat diagram view (Figure 7), you select the element that you're interested in from the drop-down list, and a Kiviat diagram will be drawn.

If any of the metrics are warnings or errors, the Kiviat diagram will display the appropriate colour.

Figure 7. Kiviat Diagram view

Metric preferences

The thresholds for each metric are controlled through the Preferences dialog. This allows you to enforce standards across projects and quickly determine whether a model is compliant with those standards. The preferences allow configuration of the colours used to highlight the different levels, as well as the thresholds for warnings and errors for each metric. You can find the preferences in Window > Preferences > Modeling > Model Metric Preferences (Figure 8).
Validation rules

To make it easier to identify problems with a model, the metrics, together with simple naming rules, are built into the validation framework. This means that when the model is validated, any metric that exceeds its warning threshold is reported. You can control which metrics are included in the validation by modifying the validation preferences in **Window > Preferences > Modeling Validation > Constraints** (Figure 9)....
Reports

The plug-in provides a reporting mechanism that you can use to analyse the results offline or to publish the metrics as HTML. Follow these steps to produce a report:

1. First, generate metrics for the model that interests you.
2. On the toolbar of the **Model Analysis** view, click the **Reporting** button.
3. You will be prompted to provide a location and then, after a short delay, the report will open, showing a page for each metric and a single page with the Kiviat diagram for each element (Figure 10).
Extending the plug-in

The Model Metric Analysis Plug-in uses the Eclipse extension point mechanism to provide each metric. Thus, it is quite simple to add your own model metrics if you prefer.

Extension point schema and example

Listing 1. Extension point schema

```xml
<element name="extension">
  <complexType>
    <sequence>
      <element ref="metric"/>
    </sequence>
    <attribute name="point" type="string" use="required">
      <annotation>
        <documentation/>
      </annotation>
    </attribute>
    <attribute name="id" type="string">
      <annotation>
        <documentation/>
      </annotation>
    </attribute>
    <attribute name="name" type="string">
      <annotation>
        <documentation/>
      </annotation>
    </attribute>
  </complexType>
</element>
```
Listing 2 is an example of a metric extension point declaration that goes into the plug-in XML file.

**Listing 2. Example of a metric extension point declaration**

```xml
<extension
    id="com.ibm.uk.views.metrics.NumberOfClassAttributes"
    name="com.ibm.uk.views.metrics.NumberOfClassAttributes"
    point="com.ibm.uk.views.modelanalysis.MetricProvider">
  <metric
    class="com.ibm.uk.views.modelanalysis.metrics.ClassAttributesMetric">
  </metric>
</extension>
```

**IMetric interface**

The IMetric interface shown in Listing 3 must be implemented and referenced in the extension point declaration.

**Listing 3. IMetric interface**

```java
public interface IMetric {

    public String getName();
    public String getShortName();
    public String getHtmlReportName();
    public boolean isApplicable(NamedElement element);

    public MetricResult[] getMetrics(NamedElement element);

    // threshold methods
    public void setDefaultPreferences();
    public int getWarningThreshold();
    public int getErrorThreshold();
    public void setWarningThreshold(int value);
    public void setErrorThreshold(int value);
}
```
Listing 4. Example of the code that implements the Number of Class Attributes metric

```java
public class ClassAttributesMetric extends AbstractMetricAndModelConstraint{

    public ClassAttributesMetric() {
        super();
    }

    public String getName() {
        return "Attributes per Class";
    }

    public boolean isApplicable(NamedElement element) {
        boolean ok = false;
        if ( element instanceof Class) {
            ok = true;
        }
        return ok;
    }

    public MetricResult[] getMetrics(NamedElement element) {
        ArrayList results = new ArrayList();
        if (isApplicable(element)) {
            if (element instanceof Class) {
                Class theClass = (Class)element;
                int attrCount = theClass.getOwnedAttributes().size();
                results.add(new MetricIntResult(this, theClass, attrCount));
            }
        }
        return (MetricResult[]) results.toArray(new MetricResult[results.size()]);
    }

    // Preferences API's
    //
    // public void setDefaultPreferences()
    //
    // IPreferenceStore prefs = ModelAnalysisPlugin.getDefault().getPreferenceStore();
    // prefs.setDefault(MetricPreferences.CLASS_ATTRIBUTE_METRIC_WARNING_THRESHOLD,10);
    // prefs.setDefault(MetricPreferences.CLASS_ATTRIBUTE_METRIC_ERROR_THRESHOLD,20);
    //
    // public int getWarningThreshold()
    //
    // IPreferenceStore prefs = ModelAnalysisPlugin.getDefault().getPreferenceStore();
    // return prefs.getInt(MetricPreferences.CLASS_ATTRIBUTE_METRIC_WARNING_THRESHOLD);
    //
    // public int getErrorThreshold()
    //
    // IPreferenceStore prefs = ModelAnalysisPlugin.getDefault().getPreferenceStore();
    // return prefs.getInt(MetricPreferences.CLASS_ATTRIBUTE_METRIC_ERROR_THRESHOLD);
    //
    // public void setWarningThreshold(int value)
    //
    // IPreferenceStore prefs = ModelAnalysisPlugin.getDefault().getPreferenceStore();
    // prefs.setValue(MetricPreferences.CLASS_ATTRIBUTE_METRIC_WARNING_THRESHOLD, value);
    //
    // public void setErrorThreshold(int value)
    //
    // IPreferenceStore prefs = ModelAnalysisPlugin.getDefault().getPreferenceStore();
}
```

Analysing models and model metrics
prefs.setValue(MetricPreferences.CLASS_ATTRIBUTE_METRIC_ERROR_THRESHOLD, value);
}

// validation methods
public IStatus validate(IValidationContext context) {
    return super.validate(context);
}

public String getHtmlReportName() {
    return "AttributesPerClass.html";
}

public String getShortName() {
    return "APC";
}

Summary

The Model Metric Analysis Plug-in helps you maintain and manage high-quality models by giving you these additional capabilities:

- Interactive analysis of models and identification of problems
- Automated discovery of problem through the addition of more than 40 metrics and rules to the model validation framework
- Review of models offline through the published model metric reports
## Downloadable resources

<table>
<thead>
<tr>
<th>Description</th>
<th>Name</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deployable RAS file</td>
<td>model_analysis_v2.0.1.ras</td>
<td>1624KB</td>
</tr>
</tbody>
</table>

© Copyright IBM Corporation 2007  
**Trademarks**  
(www.ibm.com/developerworks/ibm/trademarks/)