Generating UML models programmatically by using IBM Rational Software Architect

An introduction to the Rational Modeling Platform API

Ahmed Makady

August 26, 2008

This article shows you how to generate Unified Modeling Language (UML) models from structured text by using the IBM® Rational® Software Delivery Platform API, in particular the architecture management piece of the Rational Software Delivery Platform (Rational Modeling Platform, for short). This can save you time if you already have structured information that you would like to use for building UML models. The article gives an overview of the modeling architecture and the programming model. It also provides a detailed code example, along with a code walkthrough and an outline of the steps required to run this sample code.

Why you may need to generate UML models programmatically

The Unified Modeling Language (UML) has become a very widely adopted communication standard for software development projects around the world. In software projects, UML models are currently used to describe and communicate about software artifacts, starting from requirements through application architecture, analysis, and design, to development, deployment and maintenance.

However, in many situations, a project has information assets described in text format, and those assets need to be represented in UML to comply with best practices and to communicate with the rest of the project team.

For example, business process analysts may have business process descriptions in text that need to be converted to UML activity diagrams before delivery to the application architecture and design team to use while designing the application. UML activity diagrams are much clearer and easier for architects and designers. It may take a long time and significant effort for business analysts to go through the business process descriptions and create activity diagrams for each business process. Having a tool that automates generation of the models from text descriptions is of great help. The Rational Modeling Platform API provides the capability to create such a tool through simple Java™ code.
Introduction to the Rational Modeling Platform architecture and programming model

The Rational Modeling Platform is the foundation upon which Rational modeling solutions are based. Its most visible feature consists of a UML modeler with modeling editors, views, and tools that are built by using the various services offered by the platform. This platform is based on Eclipse technology, which is an open source tool integration platform. Rational modeling solutions include Eclipse.

Rational Modeling Platform architecture

The Rational Modeling Platform covers a wide set of technologies that can be divided into two groups: the Rational modeling components and the Eclipse components. What follows is a brief overview of each component that plays a role in this programming model.

Figure 1. Rational Modeling Platform architecture diagram

Eclipse components

- **Eclipse platform.** Central to the Eclipse technology is the Eclipse platform. This component includes frameworks to build user interfaces, manage resources, coordinate builders, integrate debuggers, and much more. The Eclipse platform is responsible for all of the basic frameworks required by an integrated development environment. IBM Rational modeling solutions make great use of the extensibility offered by this platform.

- **Eclipse Modeling Framework (EMF).** EMF is a model integration technology that serves as the backbone to IBM Rational modeling solutions.

- **UML 2.** UML 2 is an EMF-based implementation of the UML 2.1 Superstructure specification. UML is a specification that defines a graphical language for visualizing, specifying, constructing, and documenting the artifacts of distributed object systems. The Rational Modeling Platform uses this implementation for its UML 2.1 functionality.
• **EMFT Transactions.** EMFT stands for Eclipse Modeling Framework Technology. This framework contains several technologies that complement the manipulation of EMF-based models. The EMFT Transactions component provides a framework for model management (accessing and modifying models). The Rational Modeling Platform uses EMFT Transactions for all model management.

• **EMFT OCL.** EMFT OCL is an EMF-based implementation of the OCL 2.0 specification. Object Constraint Language (OCL) is a formal language used to describe rules that apply to UML models. The Rational Modeling Platform uses EMFT OCL to evaluate OCL expressions for the models that it manages, as well as to validate user input during code completion.

• **EMFT Validation.** EMFT Validation is an extensible validation framework for EMF instance models. The Rational Modeling Platform uses EMFT Validation to constrain the models that it manages. In particular, it uses the framework to constrain UML 2 models according to the official specification.

• **EMFT Query.** EMFT Query is a framework to help traversal of EMF model instances. It provides an SQL-inspired set of classes to formulate queries, as well as a rich condition class hierarchy. OCL expressions are supported to act as filters during model traversal. The Rational Modeling Platform uses EMFT Query to perform searches within the models that it manages.

• **Graphical Editing Framework (GEF).** GEF provides the basis for the creation of graphical editors. It includes a model-view-controller (MVC)-based framework, as well as a rendering subsystem. The Rational Modeling Platform uses GEF indirectly for its diagram implementation through the GMF (Graphical Modeling Framework) component.

• **Graphical Modeling Framework (GMF).** GMF is a framework to create graphical modeling editors. It consists of two main parts: a runtime component and a tooling component. The runtime part provides the infrastructure required to support extensible and feature-rich modeling diagram editors based on GEF and EMF. The tooling component enables the generation of such editors from various input models. The Rational Modeling Platform uses the GMF runtime element extensively.

### Rational Modeling Platform components

Rational Modeling Platform components span three layers: UML Modeler, UML Modeling layer, and Domain Modeling layer.

• **UML Modeler.** The UML Modeler layer includes the classes and interfaces required to manage the platform's UML modeling environment. The `UMLModeler` class is the entry point to control the lifecycle of the modeler's UML models and profiles. In addition, the class provides access to various classes required to interact with UML models.

• **UML Modeling layer.** The UML Modeling layer includes helper components to work with UML models and diagrams. For example, it includes helpers to find UML objects in models,
helpers to use UML element types (for example, activity nodes, control nodes, annotations), and helpers to find, create, customize, or analyze diagrams.

- **Domain Modeling layer.** The Domain Modeling layer includes various services useful for the production of arbitrary EMF-based modeling editors.

### Rational Modeling Platform API terms and concepts

To get started using the Rational Modeling Platform API, it is important to understand how to manage models and diagrams. The following list includes important terms and concepts that you need to know.

- **Rational Modeling Platform models.** Models managed by the Rational Modeling Platform are instances of EMF models. As an example, every UML model in the Rational Modeling Platform is an instance of an EMF-based UML metamodel available in UML 2's `org.eclipse.uml2.uml` plug-in. This plug-in is an implementation of the official UML 2.0 specification. UML models are read and modified by using the Java interfaces. Model management is done mainly through the `UMLModeler` Java class. This is a utility class that exposes model and profile lifecycle operations and provides access to the modeling platform.

- **Rational Modeling Platform editing domains.** EMF introduced the concept of an editing domain to control the lifecycle of models. An editing domain manages a self-contained set of interrelated EMF models and the commands that modify them. The editing domain used by the UML modeler for the Rational Modeling Platform is accessed through the `UMLModeler.getEditingDomain()` method.

- **Commands.** A command is responsible for the modification of a model. A command is represented by the `Command` Java interface.

- **Command stacks.** A command stack is a feature associated with an editing domain. The command stack contains commands that are executed to modify the models controlled by the editing domain. A command stack is represented by the `CommandStack` Java interface.

- **Rational Modeling Platform diagrams.** Rational Modeling Platform diagrams are based on GMF notation models. A GMF notation model provides the basic objects to describe diagrams. It defines nodes, edges, styles, and other graphical information that is required by the framework to serialize diagrams. In the Rational Modeling Platform, diagrams are added as annotations to UML model elements and are persisted in UML modeler files (.emx files). The `IUMLDiagramHelper` Java interface is used for manipulating Rational Modeling Platform diagrams.

### Sample code objective and walkthrough

The sample Java code that follows demonstrates use of the Rational Modeling Platform API. The code is presented through a Java pluglet in IBM® Rational® Software Architect. Pluglets are small Java applications that are used to make minor extensions to the workbench in a simple and straight-forward way. Pluglets are written in Java and reside in a pluglet project. The pluglet can be
tested in the same instance of the workbench, just like any other Java application. Pluglet authors can use the Java development environment, and they have access to the Rational Modeling Platform application programming interfaces (APIs).

The sample code reads business process steps from a text file and creates a UML activity diagram skeleton (one activity node for each process step) for this process in Rational Software Architect. To create this diagram, the code uses the Rational Modeling Platform API for manipulating Rational Modeling Platform models, diagrams, and model elements.

Sample code walkthrough

For better insight into the code, it is advisable that you take a look at the UML sequence diagram for the pluglet code. This diagram is available as a JPG image as part of the article material attached in the Download section.

The pluglet execution entry point is the method `plugletmain()` of the pluglet class: `GenerateActivityDiagram`. The code starts by retrieving the Rational Modeling Platform UML Modeler editing domain. As mentioned previously, the editing domains are used for manipulating Rational Modeling Platform models. The line of code in Listing 1 retrieves the modeler editing domain:

**Listing 1. Retrieve UML Modeler editing domain**

```java
TransactionalEditingDomain editDomain = UMLModeler.getEditingDomain();
```

The next step is model manipulation. According to the Rational Modeling Platform programming model, this is done through defining and running commands on the command stack associated with the editing domain. The simplest way to create a command is to extend the `RecordingCommand` class and implement the `doExecute()` abstract method to perform the required model changes. The code fragment in Listing 2 creates a new `RecordingCommand`, implements the `doExecute()` method, and actually executes the command by calling the `execute()` method on the command stack associated with the UML Modeler editing domain.

**Listing 2. Retrieve a specific UML model and modify it**

```java
editDomain.getCommandStack().execute(new RecordingCommand(editDomain, undoLabel) {
    protected void doExecute() {
        Collection models = UMLModeler.getOpenedModels();
        for (Iterator iter = models.iterator(); iter.hasNext()) {
            Model model = (Model) iter.next();
            out.println(model.getName());
            if (model == null )
                out.println("Could not open model");
            else{
                //Start operating on models having the name "*Test*
                if (model.getName().indexOf("Test") != -1){
                    createProcessActivityDiagram(model);
                }
            }
        }
    }
});
```
The `doExecute()` method in the previous code retrieves all of the models currently open in the Rational Modeling Platform environment (Rational Software Architect). Next, it iterates on these models and, for each model that contains the string "Test" as part of the name, it executes the `createProcessActivityDiagram()` method, passing the model instance to be manipulated as a parameter. This method reads sample business process steps from an external text file and adds a new activity diagram to the model that was passed as a parameter. The newly created activity diagram contains one node for each business process step read from the text file. Following is a detailed explanation for the `createProcessActivityDiagram()` method.

The `createProcessActivityDiagram()` method does all of the model manipulation work. First, it adds a new UML package, `package1`, to the model being processed. Next, it adds a new UML activity, `Activity1`, to the newly created package, and then it creates a new UML activity diagram, `ActivityDiagram1`, and adds it to the UML activity. Then, process steps are read from the text file and, for each step, an action node is added to `Activity1`. At the end, the nodes created for the actions on the diagram are adjusted for display by using the `IUMLDiagramHelper` interface method `layoutNodes()`. See the code fragment in Listing 3.

**Listing 3. Modifying the UML model in Rational Software Architect to add a package, an activity, a diagram, and action nodes**

```java
//Create a package to hold the activity diagram
Package aPackage = addPackage(model, "package1");

//Add an activity "Activity1" to the package
Activity activity1 = addActivity(aPackage, "Activity1");

//Add an activity diagram to the activity
Diagram diagram = addActivityDiagram(activity1, "ActivityDiagram1");
...

//Read process steps from a file...
in = new BufferedReader(new FileReader("...")));...

while((actionName = in.readLine())!=null){
    //Add an action to the activity
    addAction(activity1, actionName);
}
...

//Adjust diagram layout
UMLModeler.getUMLDiagramHelper().layoutNodes(diagram.getChildren(), "DEFAULT");
...
```

For more details on the methods `addPackage()`, `addActivity()`, `addActivityDiagram()`, and `addAction()`, read the Rational Modeling Platform API documentation (see Related topics). The full code is available here (see Downloadable resources).
Running the sample code

Prerequisites

This code has been developed and tested with Rational Software Architect V7.0.0. Make sure that the following extensibility features are installed with Rational Software Architect before you run the code:

- Pluglets
- Plug-in Development Environment
- Eclipse Technology Extensibility
- Modeling Extensibility

To run the sample code, follow these steps:

1. Switch to the Modeling perspective in Rational Software Architect.
2. Import the ModelGenProj project into Rational Software Architect:
   A. Select File > Import.
   B. In the Import dialog box, select Other > Project Interchange and then click Next.
   C. In the Import Project Interchange Contents dialog box, point to the file ModelGenProj.zip, check the ModelGenProj entry, and click Finish.
3. Open the GenerateActivityDiagram.java file, which is located in the com.gbm.rational.support package. Make sure that the path to the process steps file defined in the pluglet code, steps.txt, points to the right location. This location is defined in the line of code shown in Listing 4. You can either modify the file location in the code or create a folder structure on your file system that complies with the path referenced in the code.

Listing 4. Specifying the input file path

```java
in = new BufferedReader(new FileReader("..."));
```

4. Open the model named Test by double-clicking it in the Rational Software Architect Project Explorer under ModelGenProj > Models > Test. Make sure that this is the only model that is open in Rational Software Architect, because the pluglet code operates on any open model with the string "Test" as part of the name.
5. In the Project Explorer, right-click the GenerateActivityDiagram.java file, and select Run As > Pluglet.
6. After the code runs, examine your UML model. You'll find a new package created, together with a UML activity diagram created that represents the process steps defined in the steps.txt file. This file contains a list of the process steps to be modeled, each in a separate line. To view the UML diagram created, from Project Explorer, navigate to ModelGenProj > Diagrams > Test and double-click the UML diagram package1: Activity1: ActivityDiagram1.

Acknowledgement

The author thanks Amr Yassin for reviewing this article.
## Downloadable resources

<table>
<thead>
<tr>
<th>Description</th>
<th>Name</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample code and UML sequence diagram</td>
<td>Model-gen-proj.zip</td>
<td>47KB</td>
</tr>
</tbody>
</table>
Related topics

- **Getting Started with UML2**, on the Eclipse.org Web site, describes how to get started with the UML2 plug-ins for Eclipse. In particular, it gives an overview of how to create models (and their contents) both programmatically and by using the sample UML editor.
- **UML Resource Page**: Find resources, tutorials, specifications and information about the Unified Modeling Language.
- Download a **trial version of IBM Rational Software Architect**.
- Download **IBM product evaluation**.

© Copyright IBM Corporation 2008

**Trademarks**
(www.ibm.com/developerworks/ibm/trademarks/)