Reverse engineering UML class and sequence diagrams from Java code with IBM Rational Software Architect

Three techniques to overcome limitations

Skill Level: Intermediate

Fenglian Xu (axufengli@uk.ibm.com)
Software Developer
IBM

Alex Wood (wooda@uk.ibm.com)
Software Developer
IBM

10 Jun 2008

This article is for software architects, designers, and developers who want to use IBM® Rational® Software Architect to reverse engineer UML class and sequence diagrams from Java™ source code. Reverse engineering is often used to retrieve missing design documentation from existing source code in an abstract model UML format for studying both the static structure and dynamic behavior of a system and for expanding the new features to the product. The authors explain limitations of reverse engineering with Rational Software Architect and describe techniques to overcome them. You will benefit from using these technical tips and tricks to identify components and generate high-level abstractions as UML class and sequence diagrams from Java classes.

The Unified Modeling Language (UML) is well-known by all software architects, developers, and testers. It is used for documenting use cases, class diagrams, sequence diagrams, and the other diagrams. There are many software tools that help software engineers to accomplish this either by forward engineering or reverse engineering.
• **Forward engineering** is a traditional process of moving high-level abstracts and logical, implementation-independent designs to the physical implementation of a system.

• **Reverse engineering** is a process of analysing an existing system to identify its components and their interrelationships and to create representations of the system at a high level of abstraction. In most cases, reverse engineering is used to retrieve missing design documents from the existing source code in an abstract model UML format for studying both the static structure and dynamic behaviour of a system.

**Nature of the problems with class and sequence diagrams**

IBM® Rational® Software Architect is widely used by many industries because it provides many features to support reverse engineering. The problem is that when you reverse engineer UML class and sequence diagrams from Java™ code, Rational Software Architect does not automatically produce useful class and sequence diagrams. However there are techniques to improve the output from Rational Software Architect. This article demonstrates how to identify components and generate high-level abstractions of UML class and sequence diagrams from Java™ code by using the technical tips and tricks explained here.

With reverse engineering, it is not always easy to achieve what you expect from forward engineering. This article addresses problems that occurred during reverse engineering in these areas:

• Discovering abstraction classes and identify their hierarchical structures
• Producing high-level abstraction class diagrams with aggregation and association relationships
• Creating sequence diagrams

The sections that follow provide solutions to each problem and demonstrate how to produce meaningful class and sequence diagrams. The examples show how to identify the inheritance trees and components of a system in the source code of a given Java project and how to generate high-level abstractions of UML class diagrams and sequence diagrams.

**Note:**
The examples in this article were generated in Rational Software Architect Version 7.0.

**Identify an inheritance tree UML class diagram**
Inheritance is a common object-oriented pattern that allows a group of classes to share common states and behaviours so that subclasses can inherit the common states and behaviours from their superclasses. Discovering the entire inheritance tree structure from an existing system is particularly useful, because it helps you uncover what the top-level class is and what subclasses are in the tree. Furthermore, you can identify what common states and behaviours are within a tree and how the common behaviours are implemented. You can use Rational Software Architect for this discovery process in these three ways:

- Discover abstractions from a workspace or a working set
- Show an abstraction class diagram by selecting a class from the abstraction list
- Explore the tree structure in a browser diagram and you will see a list of abstract classes shown under Abstraction.

The first step is automatically retrieving the top-level abstraction classes in an existing system. You can use these classes as entry points to discover the classes in the inheritance tree. To do this, you need follow these steps:

2. Under Object-oriented Patterns, right-click Abstraction and then click Discover Architecture (see Figure 1).

This will reveal the architecture for an entire workspace.

**Figure 1. Discover the architecture for an entire workspace**

![Figure 1](image)

Figure 2 shows the rest of the steps required to produce a tree structure of an abstraction class diagram:

3. Bring up the context menu by right clicking on the Car class under
Abstraction.

4. Show the Car class diagram on the right panel by selecting **Show Diagram** from the menu.

5. Bring up the context menu by selecting and right clicking on the Car class diagram on the right panel.

6. Generate the tree structure of the abstract class diagram by selecting the **Explore in Browse Diagram** from the context menu.

As the result of step 6, you will see the tree structure of the Car class diagram on the right panel.

Figure 2. Discover abstractions from entire workspace

The result has these limitations:

- The siblings and grandchildren of the discovered class in the tree may be missed.
The classes except for the discovered abstraction class do not have attributes and operations.

Additional steps are needed to achieve the required tree structure generated by Step 6. You need to increase the degree of separation (see Figure 3), which controls the level of expansion from the discovered class.

Figure 3. Change degree of separation

The default degree is 1, and this is why some grandchildren are missed in the inheritance tree. In this example, the degree of separation has been increased to 2.

The second problem is that there are no attributes and operations for the classes except for the discovered class in the tree. This prevents users from studying the existing common patterns for reuse.

The following example demonstrates how to identify an entire inheritance tree with optional attributes and operations:

3. Load a Java project into Rational Software Architect.

4. Switch to the **Diagram Navigation** view and discover the abstractions from the workspace as described in the previous steps.

5. Find an abstraction class from Step 2 in which you are interested.

6. Discover the hierarchical tree structure classes by finding the class in the Model Explorer, double-click to open the class in the editor, and press **F4** to open the hierarchy tree. Ensure that the **type hierarchy** is selected.

7. Right-click on one class and change it to a visual class diagram by selecting **Visualize > Add to New Diagram File > Class Diagram**, as shown in Figure 4.

8. Add the rest of the classes to the current diagram by right-clicking and selecting **Visualize > Add to Current Diagram**.

Figure 4. Visualize class into new class diagram
Figure 5 demonstrates the process of generating an inheritance tree class diagram:

9. Open and press **F4** to show the class hierarchy.
10. Select each class and add it to the class diagram.
11. Review the completed diagram on the right.

**Figure 5. A mechanism to generate an inheritance tree class diagram**

The class diagram is generated with the Rational Software Architect default format. There are several useful modifications that you can make to the visual representation of the diagram. For example, you can modify the connections routing.
style to use the tree style routing, and you can right-click in the workspace to bring up a context menu and then click Arrange all. The class diagram now looks much better than the one automatically generated, as Figure 6 shows.

**Figure 6. Inheritance tree diagram with attributes and the tree style routing connections**

![Inheritance tree diagram]

The classes in Figure 6 show both attributes and operations. The benefit of showing the attributes and operations is that you can study the common states and behaviours further to discover what has been implemented in the existing system for reuse.
Generate a high-level UML class diagram

Rational Software Architect enables you to generate class diagrams by selecting multiple Java files from a Java project:

1. In the Model Explorer, use **Visualize** to add them to a new diagram or the current class diagram.

If multiple classes have been added in the current diagram, then the relationships among these classes are also shown.

Figure 7 is an example of a class diagram that was automatically generated from the Java code in a package.

**Figure 7. An auto-generated class diagram**

As Figure 7 shows, you can select multiple Java files from the Model Explorer to visualize them in a new class diagram. If you want to add more classes, you can select more Java source code to visualize in the current class diagram. This diagram
shows the classes included in the project and their basic relationships. It is useful to automatically discover the UML classes in the project, but the auto-discovered relationships are less useful here.

In Figure 7, almost every relationship is a use apart from the inheritance. The use relationship is too general to give useful design information. The more specific relationships of aggregation and composition are hidden, even when all of the relationships are turned on for this diagram. The aggregation is represented as one-to-many relationships when a class contains many items of another class, and the composition is used for describing a one-to-one relationship when a class contains only one instance of another class. This high-level abstraction represents more accurate relationships among the classes and provides useful information for the implementation of this design. This class diagram is not as useful without the details of the abstract relationships.

Here, we have experienced and explored a semi-automated method for generating a high-level abstraction of a UML class diagram. The UML classes are discovered with the same technique as before, and the relationships among classes are created manually. The high-level abstraction is based on the knowledge acquired from studying the existing source code.

The example shown in Figure 8 demonstrates how to apply this method to create the high-level UML class diagram.

**Figure 8. Create a blank model by using the UML model wizard**
To generate a high-level class diagram, you need to create a blank model first:
2. Create a new blank model by using the steps shown in Figure 8:
   A. Under File Types, select **UML Modeling**.
   B. Under Templates, select **Blank Model**.
   C. In the File Name field, type **Blank Model**.
   D. For Destination folder, type **example**.
   E. The Default diagram check box for "Create a default diagram in the new model" should be checked.
   F. For Default diagram type, select **Freeform Diagram**.
   G. Click **Finish**.

The next step is to harvest the selected classes from the auto-generated class diagram. Harvesting in Rational Software Architect enables you to copy a class from one class diagram and paste it into another class diagram, which must be within a blank model. If you paste the harvested class into the same class diagram or into another class diagram that is outside of a blank model, the class attributes and functions are invisible.

3. Harvest the selected Java classes from the auto-generated class diagram by using the steps shown in Figure 9:
   A. Select the **FuelTank**, **Engine**, **Passenger** and **Car** classes from the classdiagram2.dnx.
   B. Right click on one of the selected classes to bring up a context menu.
   C. Select the **Harvest** menu

4. Paste the harvested classes into the separated class diagram that you created in Step 2.

5. Create relationships with aggregation or composition between the classes.

**Figure 9. Harvest classes from a class diagram**
Next, create an association relationship between the classes that enables you to choose either an aggregation or a composition relationship. An example of a high-level class diagram is shown in Figure 10.

**Figure 10. A high-level abstraction of a UML class diagram generated by the semi-automatic method**
By comparing Figure 10 with Figure 9, you can tell that the class relationships have been shown accurately by the semi-automated method. This diagram can be used for an implementation independent design document or for exploring any further improvement to the existing system.

**Important:**
Without harvesting, the aggregation and composition relationships cannot be used in Rational Software Architect

**Generate a sequence diagram**

The sequence diagram is the most popular UML artifact for dynamic modeling. It focuses on identifying the behaviour of a system. A sequence diagram is typically used for modeling use cases, to show the logic of both the methods and the services of a system.
Rational Software Architect does not create a sequence diagram automatically from the Java code. The following steps show you how to create one:

1. Create a blank model.
2. Create a sequence diagram:
   A. Right-click on a **Blank Model**.
   B. From the drop-down (context) menus, select **Add Diagram** and then select **Sequence Diagram** (see Figure 11).
3. Add the classes to the sequence diagram.
4. Make a sequence of method invocation between two classes.
5. Save the sequence diagram.

**Figure 11. Create a sequence diagram**

When you have finished creating a sequence diagram, a sequence file is created under the **Collaboration: Interaction** tab. You can add the class from the Java code to the sequence diagram. Both are shown in Figure 12.
The main workspace in Figure 12 shows an example of a sequence diagram.

**Figure 12. Sequence diagram created from Java source files**

A method invocation is represented by a message sent from a caller to a callee. The callee is the method owner that receives a message from a method caller. The message can be either a one-way or a two-way message. A sequence diagram consists of a sequence of method invocations among a group of method owners and an initial invoker. The first invocation must be started from an actor that initialises the first method invocation.
Summary

This article demonstrated how to use reverse engineering to create UML class and sequence diagrams from Java code by using Rational Software Architect v7.0. The hierarchical class diagram demonstrates a way to discover an entire class inheritance relationship within a project or a work set. This helps developers expand and add new functions to an existing system. The high-level class diagram clearly shows aggregation or composition relationships between classes, which is helpful to developers to extend or modify an existing design. This is particularly powerful for developers to explore a large system. The sequence diagram shows the dynamic method invocations among classes for performing certain tasks. This provides clear runtime interactions within the system.

Acknowledgements

The authors thank Philip Norton and Noel Rooney for their reviews and feedback on this article.
Resources

Learn

• Read the white paper titled Reverse Engineering of UML Class and Sequence Diagrams from Java by Philip Hartman (2006).

• Learn more by reading the IEEE article called Reverse Engineering is Reverse Forward Engineering by Ira D. Baxter and Michael Mehlich (1997).

• Visit the Rational software area on developerWorks for technical resources and best practices for Rational Software Delivery Platform products.

• Explore Rational computer-based, Web-based, and instructor-led online courses. Hone your skills and learn more about Rational tools with these courses, which range from introductory to advanced. The courses on this catalog are available for purchase through computer-based training or Web-based training. Additionally, some "Getting Started" courses are available free of charge.

• Subscribe to the Rational Edge newsletter for articles on the concepts behind effective software development.

• Subscribe to the IBM developerWorks newsletter, a weekly update on the best of developerWorks tutorials, articles, downloads, community activities, webcasts and events.

• Browse the technology bookstore for books on these and other technical topics.

Get products and technologies

• Download trial versions of IBM Rational software.

• Download these IBM product evaluation versions and get your hands on application development tools and middleware products from DB2®, Lotus®, Tivoli®, and WebSphere®.

Discuss

• Check out developerWorks blogs and get involved in the developerWorks community.


About the authors

Fenglian Xu
Fenglian Xu works as a software developer for the IBM WebSphere Enterprise Service Bus and WebSphere Process Server development team in the IBM Hursley Lab in the United Kingdom. Her additional expertise includes WebSphere MQ, Java Message Service (JMS), WebSphere Message Bindings and Service Oriented Architecture (SOA). She has worked in various middleware and application IT companies, and she often uses UML tools for writing design documents. She received her BSc in Applied Math and Software Engineering from Xian Jiaotong University in China in 1989 and a PhD in Computer Science from the University of Southampton in 1998.

Alex Wood
Alex Wood works at IBM Hursley Lab in England as a software developer for the IBM WebSphere Business Integration suite of products. He has extensive experience in development on many of the WebSphere products, including WebSphere MQ, WebSphere Message Broker, WebSphere Enterprise Service Bus, and WebSphere Process Server. He received a BSc in Physics with Astrophysics from Birmingham University in the UK in 1998.