Hello World: Rational Software Architect V7

Design and implement a simple phone book application (updated)

Skill Level: Intermediate

Tinny Ng (tng@ca.ibm.com)
Advisory Software Developer
IBM Toronto

20 Feb 2007

This first tutorial in the Hello World series has been updated for Rational Software Architect V7. Hands-on exercises walk you through every step of designing and implementing a service-oriented application using Rational® Software Architect V7, including using UML diagrams to design and model an application, publishing the model information to a Web page, and transforming the design to Java™ code. It concludes with an introduction to Rational Software Architect V7’s new true round-trip engineering (RTE), which makes it possible to forward- and reverse-engineer Java code, giving you a head-start on implementing your designs.

Section 1. Before you start

About this series

The Hello World series is for novice developers who want a high-level, hands-on overview of IBM software products. Each tutorial in the series provides simple exercises and step-by-step instructions to familiarize you with the components and use of a particular product. Upon completing a tutorial in the Hello World series you will know enough about the product to begin exploring and using it on your own.
About this tutorial

This first tutorial in the Hello World series has been updated for Rational Software Architect V7. The tutorial guides you step-by-step through the process of designing and developing a simple service-oriented application using Rational Software Architect V7. It includes practical exercises that show you how to use UML diagrams to design the application, publish the model information to a Web page, and transform the design to Java code. This tutorial concludes with an introduction to Rational Software Architect’s true round-trip engineering (RTE), which supports the process of forward- and reverse-engineering Java code. The original version of this tutorial for Rational Software Architect V6 is still available. For introductions to other products that play key roles in a service-oriented architecture, see the complete list of Hello World tutorials.

Prerequisites

This tutorial is for beginning application developers and requires no experience with Rational Software Architect. Rational Software Architect V7 is built on Eclipse 3.2 and includes support for JDK 5. To get the most out of the tutorial you should have some experience using the Eclipse development environment.

System requirements

To run the examples in the tutorial you must install Rational Software Architect V7 before you begin. If you want to view the demos you must also enable JavaScript in your browser and install Macromedia Flash Player 6 or higher.

Section 2. Introduction

IBM Rational Software Architect V7 is an integrated design and development tool built on top of the Eclipse platform, an open and extensible development platform that leverages industry standards. Rational Software Architect unifies three tools for implementing application architecture, design, and development:

- Rational Software Modeler V7: A modeling tool that lets you visually
model systems and applications using Unified Modeling Language (UML) notation.

- Rational Systems Developer V7: A modeling tool that supports model-driven development for software products and systems development.
- Rational Application Developer V7: A development tool that facilitates J2EE development, XML development, Web services development, and more.

Software architects and senior developers within a development team can use Rational Software Architect to specify and maintain all aspects of an application’s software architecture. Support for UML 2 makes it possible to capture and communicate all aspects of an application architecture using a standard notation. Patterns and transformations ease the process of defining and implementing applications.

Table 1 highlights some of the key features and benefits of Rational Software Architect V7; the first three features are illustrated by exercises later in the tutorial. See Resources for links to more product information.

### Table 1. Key features of Rational Software Architect V7

<table>
<thead>
<tr>
<th>Features</th>
<th>Benefits</th>
</tr>
</thead>
<tbody>
<tr>
<td>UML 2 modeling support for analysis and design using Use Case, Class, Object, Sequence, Activity, Composite Structure, State Machine, Communication, Component, and Deployment diagrams</td>
<td>Allows you to capture and communicate all aspects of an application architecture using a standard notation that is recognized by many different stakeholders.</td>
</tr>
<tr>
<td>Generates HTML, PDF, and XML reports from UML designs.</td>
<td>Allows you to create reports and documentation that can be reviewed by team members or other stakeholders.</td>
</tr>
<tr>
<td>Uses transformations to generate Java, C++, Enterprise Java Bean, WSDL, XSD, and CORBA IDL code.</td>
<td>Automates the repeatable task of generating code from design models. Transformations can be customized to tailor code-generation patterns to an organization's needs.</td>
</tr>
<tr>
<td>UML class diagram editing for Java code, EJB components and database objects, and C++.</td>
<td>Stylized UML notation provides abstract views of Java code, EJB components, database objects, and C++, simplifying the development and understanding of new and existing applications.</td>
</tr>
<tr>
<td>Java method body visualization using UML 2 sequence diagrams.</td>
<td>UML 2 sequence diagram constructs assist you in understanding the flow of a Java method.</td>
</tr>
<tr>
<td>WS-I compliant Web services and service-oriented architectures.</td>
<td>Integrates your business applications.</td>
</tr>
</tbody>
</table>
Apply and author patterns and transforms. | Allows organizations to capture and promote “recipes” that can be used to increase the predictability and repeatability of software development. The authoring and apply capabilities support teams in developing for reuse and developing with reuse.

Asset Browser for accessing reusable assets. | Supports OMG Reusable Asset Specification and supports users in browsing repositories containing reusable assets. Repositories can be structured so that assets can be easily found.

Open API supports customizing and extending the modeling environment. UML profile creation and editing to customize the properties stored in UML models. | Organizations can develop plug-ins to customize the analysis and design tools for their environment and process. This promotes an ecosystem fostering the development of third-party extensions and integrations.

RUP configuration for software architects with context-sensitive and dynamic-process guidance. | Process guidance and user assistance is provided dynamically as the user works with the tool.

---

Section 3. How does Rational Software Architect fit into SOA?

Companies in every industry want to respond more quickly and effectively to changing market conditions. To achieve this level of business flexibility, many companies are implementing Service Oriented Architecture (SOA). SOA is an architectural style for building distributed systems that deliver application functions as services to be used by end-user applications, or for building other services. In an SOA environment each of a company’s business functions can be implemented as a service. Each service can then be integrated with other services to fulfill the company's business requirements. SOA makes it possible to create sophisticated applications and solutions swiftly and easily by assembling from new and existing services.

Getting started with SOA is easy with the IBM SOA Foundation -- an integrated, open-standards-based set of software, best practices, and patterns for Service Oriented Architecture. The software that comprises the IBM SOA Foundation supports each stage of the SOA life cycle: model, assemble, deploy, and manage. You can see a diagram of the SOA life cycle in Figure 1. Underpinning all of the life-cycle stages are governance and processes that give guidance and oversight to
the SOA project.

**Figure 1. The SOA life cycle**

Rational Software Architect is part of the IBM SOA Foundation and supports the model phase of the SOA life cycle. It is considered one of the development services of the SOA Reference Architecture, shown in Figure 2, because it gives companies the tools needed to model service-oriented applications.

**Figure 2. The SOA Reference Architecture**
Rational Software Architect lets software architects visually model and design a flexible services architecture using the open standard UML, and automatically apply design patterns for SOA from analysis and design to implementation. To further assist users with solution design in a service-oriented world, many new SOA design resources are available as Rational Software Architect plug-ins. See Resources to learn more about design resources for Rational Software Architect.

Section 4. What is UML?

The Unified Modeling Language, or UML, was first released by the Object Management Group (OMG) in 1997. The current official version is UML 2.0. UML is designed to bring together the development community with a stable and common design language that can be used to develop and build applications. It is a modeling language and is independent of any programming language.
UML provides various diagrams that enable users to capture and communicate all aspects of an application architecture using a standard notation that is recognized by many different stakeholders. There are 13 official UML 2.0 diagrams, each of which is a different view that shows different aspects of the system:

- Activity diagram
- Class diagram
- Communication diagram
- Component diagram
- Composite structure diagram
- Deployment diagram
- Interaction overview diagram
- Object diagram
- Package diagram
- Sequence diagram
- State machine diagram
- Timing diagram
- Use case diagram

In the following exercises you'll learn how to create a use case diagram, a class diagram, and a sequence diagram, and then publish and transform the design using Rational Software Architect. You'll also have the opportunity to use Rational Software Architect's new true round-trip engineering (RTE) to forward- and reverse-engineer the resulting application's Java code. The tutorial will conclude with an implementation example.

Section 5. Design a phone book application

The very simple phone book application shown in Figure 3 stores user-entered phone numbers to be retrieved later.

**Figure 3. The phone book application**
To begin designing the application, start up Rational Software Architect on your desktop. From Windows select **Start > Programs > IBM Software Development Platform > IBM Rational Software Architect > IBM Rational Software Architect**.

A window appears asking for the workspace directory. Select **OK** to accept the default.

Would you like to see these steps demonstrated for you?

Show me

Create a UML project

Once Rational Software Architect is up and running, create a UML project named MyPhoneBookUMLProject. The steps to do this are as follows:

1. From the workbench, select **File > New > Project**.
2. Select **UML Project** and then select **Next**.
3. Enter **MyPhoneBookUMLProject** as the project name, and select **Next**.
4. Enter **Phone Book UML Model** as the file name of the UML model, uncheck the box **Create a default diagram in the new model**, then select **Finish**.
5. Click **Yes** if you are asked if you want to open the Modeling perspective.
Section 6. Create a use case diagram

A use case diagram models the behavior of a system and helps to capture the system requirements. It identifies the interactions between the system and its actors and defines the scope of the system.

Actor
Represents a role of a user that interacts with the system. The user can be a human user, an organization, a machine, or another external system.

Use case
Describes a function that a system performs to achieve the user’s goal. A use case must yield an observable result that is of value to the user of the system.

The use cases and actors shown in a use case diagram describe what the system does and how the actors use it, but not how the system operates internally. To relate an actor and a use case, you can create an association relationship indicating the connection between the two model elements.

For the simple phone book application there is only one actor, Any User, who can carry out the following two use cases against the system:

1. **Add an entry**
   Enter a unique person name and a phone number using the provided application user interface. The system processes the entered data and stores it.

2. **Search for a phone number**
   Retrieve a phone number by entering a unique person name using the provided application user interface. The system locates the phone number and returns it to the actor.

Would you like to see these steps demonstrated for you?

Follow the steps below to create a use case diagram of the two use cases for the simple phone book application.

1. In Rational Software Architect’s Model Explorer view, right-click **Phone**
Book UML Model and select Add Diagram > Use case diagram, as shown.
Figure 4. Adding a use case diagram

2. Enter use case diagram as the name of the generated diagram to replace the default name UseCaseDiagram1. Now you can draw the use case diagram by adding various model elements from the Palette to the diagram, as shown in Figure 5.
Figure 5. Adding model elements
3. Select **Actor** in the Palette, then click anywhere in the diagram to create an actor. Name it Any User.

4. Select **Use Case** in the Palette, then click anywhere in the diagram to create a use case. Name it Add an entry.

5. Similarly, create another use case called **Search for a phone number**.

6. Select **Association** in the Palette. Draw the association relationship line from the actor Any User to the use case Add an entry to initiate a relationship between the two model elements.

7. Name the relationship use case 1.

8. Similarly, create another association relationship between the actor Any User and the use case **Search for a phone number** and name it use case 2.
Section 7. Create a class diagram

Class diagrams are the blueprints of your system. You can use class diagrams to model the objects that make up the system, to display the relationships between the objects, and to describe what those objects do and the services they provide.

You'll design the simple phone book application using the Model-View-Controller (MVC) architecture pattern, as shown in Figure 7. (See Resources to learn about MVC.) For the phone book application you'll need to create the following three classes:

- **PhoneBookModel** manages the phone book entries and captures the state of the application. Whenever the state is changed, it notifies
PhoneBookView, which should then refresh the user interface based on the state of the application.

- **PhoneBookView** manages the graphical or textual interface to the user based on the state of the application, and notifies **PhoneBookController** when an input is received.

- **PhoneBookController** controls the operation of the entire application. It changes the model state of the application and updates the data model based on user input.

The purpose of this exercise is to show you how to use Rational Software Architect when designing an application. The design itself is not the focus. You may proceed with a different design if you prefer.

**Figure 7. MVC design for the phone book application**

Now create a class diagram that reflects the design in Figure 7.

1. In Rational Software Architect's Model Explorer view, right-click **Phone Book UML Model** and select **Add Diagram > Class Diagram**.

2. Enter **Class Diagram** as the name of the generated diagram to replace the default name **ClassDiagram1**.

3. Select **Class** in the Palette, then click anywhere in the diagram to create a class. Name it **PhoneBookModel**.

4. Right-click the created class **PhoneBookModel** and select **Add UML >**
**Operation** to create an operation for this class. Name it `setState`.

5. Similarly, create the rest of the elements as shown in Table 2. The operation names are chosen without assumptions regarding the programming language used.

**Table 2. Class information**

<table>
<thead>
<tr>
<th>Class</th>
<th>Operation(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PhoneBookModel</td>
<td><code>addAnEntry</code> <code>searchPhoneNumber</code> <code>getSearchResult</code> <code>getState</code></td>
</tr>
<tr>
<td>PhoneBookView</td>
<td><code>stateHasChanged</code> <code>changeView</code> <code>getUserInput</code></td>
</tr>
<tr>
<td>PhoneBookController</td>
<td><code>userHasInput</code> <code>start</code></td>
</tr>
</tbody>
</table>

6. Next you'll create some associations to relate these three classes. As shown in Figure 8, click the arrow that appears next to Association from the Palette and select **Directed Association**.

**Figure 8. Selecting Directed Association from the Palette**

7. Draw the association relationship line from the class `PhoneBookController` to `PhoneBookModel` *(the order is important)* to initiate a directed association relationship between these two classes. A **directed association relationship** means the first class is aware of the
second one, but not the other way around.

8. Name this directed association relationship **controllerHasModel**.

9. Now that you’ve done it once, create a couple more relationships. This time create association relationships **without direction**, which means the two connected classes are mutually aware of each other:

   - Create an association relationship between the class `PhoneBookModel` and `PhoneBookView` and name it `modelView`.
   - Create an association relationship between the class `PhoneBookView` and `PhoneBookController` and name it `viewController`.

10. The complete class diagram should look similar to Figure 9. Select **Ctrl-S** to save the diagram.

**Figure 9. A complete class diagram for the phone book application**

---

**Section 8. Create a sequence diagram**
A sequence diagram in UML shows the chronological sequence of messages between instances in an interaction. It consists of an interaction that is represented by lifelines and the messages that they exchange during the interaction, also known as message lines.

In this section you will realize the use case "Search for a phone number" and show the associated interaction using a sequence diagram. The use case starts with the actor Any User, who makes use of the interface provided by PhoneBookView to request a search. PhoneBookView notifies PhoneBookController about the user's request. PhoneBookController then updates the data model that is stored in PhoneBookModel. Because there is a change of the model, PhoneBookModel notifies PhoneBookView, which should then refresh the user interface to reflect the latest state of the application.

Would you like to see these steps demonstrated for you?

Show me

1. In Rational Software Architect's Model Explorer view, right-click PhoneBook UML Model and select Add Diagram > Sequence Diagram.

2. Enter Sequence Diagram as the name of the generated diagram to replace the default name SequenceDiagram1.

3. Drag the actor Any User from the Model Explorer view to the diagram to create an instance of the actor, as shown in Figure 10. Similarly, create instances of PhoneBookView, PhoneBookController, and PhoneBookModel by dragging them to the diagram.

Figure 10. Dragging model elements to the sequence diagram
4. Select **Asynchronous Message** in the Palette. As shown in Figure 11, click the line under **any User: Any User**, hold the mouse click, and then release the mouse click at the line under **phoneBookView:PhoneBookView**.

Figure 11. Creating a message line
5. Select the operation `PhoneBookView::getUserInput()` from the drop-down list, as shown.

**Figure 12. Selecting the operation for a message line**

6. Similarly, create the following asynchronous message lines shown in Table 3. To create an asynchronous message to call itself, simply click the instance bar directly without any dragging.
Table 3. Asynchronous message lines for the sequence diagram

<table>
<thead>
<tr>
<th>From instance</th>
<th>To instance</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>phoneBookView</td>
<td>phoneBookController</td>
<td>PhoneBookController::userHasInput()</td>
</tr>
<tr>
<td>phoneBookController</td>
<td>phoneBookModel</td>
<td>PhoneBookModel::searchPhoneNumber()</td>
</tr>
<tr>
<td>phoneBookController</td>
<td>phoneBookModel</td>
<td>PhoneBookModel::setState()</td>
</tr>
<tr>
<td>phoneBookModel</td>
<td>phoneBookView</td>
<td>PhoneBookView::stateHasChanged()</td>
</tr>
<tr>
<td>phoneBookView</td>
<td>phoneBookModel</td>
<td>PhoneBookModel::getSearchResult()</td>
</tr>
<tr>
<td>phoneBookView</td>
<td>phoneBookView</td>
<td>PhoneBookView::changeView()</td>
</tr>
</tbody>
</table>

7. The complete sequence diagram should look like Figure 13. Select File > Save All to save everything.

Figure 13. The complete sequence diagram

Section 9. Publishing the design

Publishing the model information for your application design allows you to share it with stakeholders who do not have a modeling tool. In this section you will learn how
to publish your design to a Web page.

Would you like to see these steps demonstrated for you?

Show me

1. Select **Phone Book UML Model** in Rational Software Architect's Model Explorer view.

2. Select **Modeling > Publish > Web**, as shown in Figure 14.

   **Figure 14. Publishing a design to a Web page**

3. Specify the target location of the generated HTML files, for example C:\HelloWorldSeries\RSA_Web, and select **OK**. The model is then published into HTML files that are stored in the specified location.

4. Open the file C:\HelloWorldSeries\RSA_Web\index.html, as shown in Figure 15, using any Web browser.

   **Figure 15. A published Web page**
5. Select the link **Phone Book UML Model**.

6. Navigate the published model by clicking the links for **Elements** and **Diagrams**, as shown in Figure 16.

**Figure 16. Navigating the published Phone Book UML Model**
Section 10. Transformation and true round-trip engineering

Transformation is a key feature of Rational Software Architect, allowing you to easily transform your designs from UML to EJB components, WSDL, Java code, and so on. In Rational Software Architect V7, the transformation function has been extended to include support for true round-trip engineering (RTE), giving you even more of a head-start in implementing your designs.

Before you can run a transformation, you must create a transformation configuration. A transformation configuration is an instance of a transformation that includes information that all transformations use, such as a unique name, the source, and the
target of the transformation. A transformation configuration can also include properties that are specific to a given transformation. When you run a transformation, it uses the information provided in the transformation configuration.

The Rational Software Architect V7 tutorial gallery includes a tutorial called "Concurrently develop UML models and Java code using transformations," which shows you how to create a UML-to-Java transformation configuration that applies the **Replace UML elements** option. When you run a UML-to-Java transformation, the elements in the UML model are replaced with visual representations of the corresponding generated Java classes. This replacement enables you to make changes to the source code from the model and vice versa. The changes that you make to the model are reflected automatically in the corresponding Java file and the changes that you make to the Java classes are reflected automatically in the corresponding model element.

**Figure 17. A screen shot from the UML-to-Java transformation tutorial**
In this tutorial, you will learn how to transform your UML design into Java \textit{without} using the \textbf{Replace UML elements} option. Instead, you'll first transform the model into Java code, make a small change in the generated Java file, and then transform the Java class into another UML model. You'll then compare the original UML model with the reverse-engineered UML model to see the difference.
Transform UML to Java

First, create a transformation configuration for the UML-to-Java transformation.

Would you like to see these steps demonstrated for you?

Show me


2. Select Modeling > Transform > New Configuration.

3. Enter the name PhoneBookUML2Java and select IBM Rational > UML to Java V5.0, as shown in Figure 18. Click Next.

Figure 18. Creating a transformation configuration
4. Select **MyPhoneBookUML > Models > Phone Book UML Model** as the source.

5. Click **Create new Target Container...** to create a Java project for storing the generated files.
6. In the New Java Project panel, enter PhoneBookJavaProject as the Java project name. Click **Finish**.

7. In the Source and Target panel, select the newly created Java project **PhoneBookJavaProject** as your target, as shown in Figure 19.

![Image: Selecting a source and target for transformation](image)

**Figure 19. Selecting a source and target for transformation**

8. Click **Next**.

9. Unselect **Generate getter and setter** so that no getters and setters will be generated, as shown in Figure 20.

![Image: Unselecting getter and setter generation](image)

**Figure 20. Unselecting getter and setter generation**
You have now created a transformation configuration called PhoneBookUML2Java.tc. You have also created a PhoneBookJavaProject, but no Java files are yet being generated. Your next step is to transform the UML model to Java code.

1. Right-click the transformation configuration file PhoneBookUML2Java.tc.

2. Select Transform > UML to Java V5.0 to transform the UML model to Java code, as shown in Figure 21.

3. Navigate the newly generated Java files to see how the UML model elements are being transformed. For example, take a look at PhoneBookController.java, shown in Figure 22. The generated class has
two attributes, phonebookmodel and phonebookview, which are generated, respectively, as a result of the directed association and association relationships you created earlier. The two operations (userHasInput and start) that you added when drawing the class diagram are also generated.

Figure 22. PhoneBookController.java

```java
public class PhoneBookController {  
    /**
     * @generated "UML to Java (com.ibm.xtools.transform.uml2
     */
    private PhoneBookModel phonebookmodel;

    /**
     * @generated "UML to Java (com.ibm.xtools.transform.uml2
     */
    private PhoneBookView phonebookview;

    /**
     * @generated "UML to Java (com.ibm.xtools.transform.uml2
     */
    public void userHasInput() {
        // TODO Auto-generated method stub
    }

    /**
     * @generated "UML to Java (com.ibm.xtools.transform.uml2
     */
    public void start() {
        // TODO Auto-generated method stub
    }
}
```

4. Make a change to PhoneBookController.java, as shown in Figure 23. You will see later how this change is reflected in the reverse-engineered UML model.

Figure 23. Adding a parameter to userHasInput()
5. Press **Ctrl S** to save all the changes.

**Transform Java to UML**

Now create another transformation configuration to transform the Java files back to a UML model.

1. Select **PhoneBookJavaProject** in Rational Software Architect’s Model Explorer view.

2. Select **Modeling > Transform > New Configuration**.

3. Enter the name **PhoneBookJava2UML** and select **IBM Rational > Java to UML**, as shown in Figure 24. Click **Next**.

**Figure 24. Creating a transformation configuration for Java-to-UML transformation**
4. Select **PhoneBookJavaProject** as the source.

5. Click **Create new Target Container...** to create a new UML model for storing the generated files.

6. In the Create UML Model panel, click **Next** to create a new UML model from the standard template.

7. Enter **Phone Book UML Reverse Model** as the file name.
8. Click **Browse** to select the existing **MyPhoneBookUMLProject** as the location for the new model file. The Create UML Model panel should look as shown in Figure 25.

**Figure 25. The Create UML Model panel**

![Create UML Model Panel](image)

9. Click **Finish**.

10. In the Source and Target panel, select the newly created model file **Phone Book UML Reverse Model** as the target, as shown in Figure 26.

**Figure 26. Selecting the source and target for transformation**

![Select Source and Target Panel](image)
11. Click **Next**.

12. Unselect **Generate getter and setter** so that no getters and setters will be generated.

13. Continue clicking **Next** to see what other options are available. Take all default options, then click **Finish**.

You have now created a transformation configuration called **PhoneBookJava2UML.tc**. The last step in this exercise is to transform the Java files to a UML model.

1. Right-click the file **PhoneBookJava2UML.tc**.

2. Select **Transform > Java to UML** to transform the Java code to a UML model, as shown in Figure 27.

   **Figure 27. Transforming Java code to UML**

3. Click **OK** to accept any pending changes.

4. The three Java classes are now transformed to three model elements
under the **Phone Book UML Reverse Model**, as shown in Figure 28.

**Figure 28. The generated UML model**

5. Create a Class Diagram in the Phone Book UML Reverse Model to visualize the generated model elements. Drag and drop the three model classes into the diagram, as shown in Figure 29.

**Figure 29. The three model classes in the class diagram**

6. The association relationships are shown as attributes in this diagram. Select those attributes, then right-click and select **Filters > Show As Association** to turn those attributes into association relationships.

7. The new parameter that you’ve added in **userHasInput** is not obvious in the diagram. Right-click the class **PhoneBookController** and select **Filters > Show Signature** to display the full signature of the operations.
8. The class diagram for your transformed UML model should look as shown in Figure 30. Compare this diagram with the class diagram you created earlier (see Figure 9) and notice the similarity. Notice also that the Association Relationship is shown as two lines in this diagram to indicate the bidirectional behavior.

Figure 30. The generated UML model

9. Press Ctrl S to save all the changes.

Section 11. Complete the implementation

Once you have generated Java files from your UML diagram, you can modify the generated files to implement your application based on the design infrastructure. In the Downloads section you'll find a sample phone book application that implements a command-line interface and stores the phone entries into a local file, but this is only one possible implementation. For example, you could create a GUI interface instead of a command-line interface, store the data with EJB components instead of to a file, or use the Observer pattern to implement the application's notification mechanism. The provided sample implementation is just one of many ways to implement the design.
In this final section you will modify the files generated from your UML diagram to implement the sample phone book application.

Would you like to see these steps demonstrated for you?

Show me

1. Copy and paste the sample implementation into the three Java files as appropriate.

2. Select File > Save All to save everything.

3. Select PhoneBookController, then select Run > Run As ... > Java Application to run the phone book implementation as a Java application.

4. Go to the Console view shown in Figure 31 and interact with the application. Validate that you can perform the application's two use cases, Add an entry and Search for a phone number. Remember: the purpose of use cases is to define the behavior of a system and capture the requirements. It is important that the implementation fulfills the requirements by working as expected.

   Figure 31. Running the sample phone book application

5. You can also run a Java to UML transformation again, using the PhoneBookJava2UML.tc transformation configuration to understand the UML model of the complete phone book application.
Figure 32. The UML model of the sample phone book application

Section 12. Summary

This tutorial has provided a high-level, hands-on introduction to Rational Software Architect V7. You have followed step-by-step exercises and demonstrations to learn how to design an application using UML diagrams, publish the model information to a Web page, transform your design to Java code, and then reverse-engineer the Java code back to a UML diagram using Rational Software Architect V7. Finally, you
implemented the simple phone book application and viewed it in the Rational Software Architect console.

See Resources to learn more about Rational Software Architect and other IBM software products covered in the Hello World tutorial series.
## Downloads

<table>
<thead>
<tr>
<th>Description</th>
<th>Name</th>
<th>Size</th>
<th>Download method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample phone book application</td>
<td>phone.zip</td>
<td>4KB</td>
<td>HTTP</td>
</tr>
</tbody>
</table>

Information about download methods
Resources

Learn

- The Hello World series of hands-on tutorials introduces IBM software products that play a critical role in implementing an SOA foundation in your enterprise.

- See the IBM Rational Software Architect home page for technical documentation, how-to articles, education, downloads, and product information about Rational Software Architect.

- Learn more about the IBM SOA Foundation -- an integrated, open-standards-based set of software, best practices, and patterns for Service Oriented Architecture.


- "UML basics: An introduction to the Unified Modeling Language" (developerWorks, June 2003) introduces the most important diagrams in UML modeling.

- Learn more about the Model-View-Controller design pattern.

Get products and technologies

- Download a free trial version of Rational Software Architect V7.

- Build your next development project with IBM trial software, available for download directly from developerWorks.

- Discover new SOA design resources available from IBM.

- Download Eclipse 3.2.

Discuss

- Participate in the discussion forum for this content.

About the author
Tinny Ng
Tinny Ng is a scenario architect at the IBM SWG Scenario Analysis Lab, whose mission is to improve the cross-brand integration capability of IBM SWG products. She has more than 10 years of experience in software development, from design to implementation, including application building, packaging, testing, and support.