Rational Software Architect

Quick Start Tutorial

Contents

1. Introduction

Rational Software Architect is a tool that enables software architects to model and design the architecture of their applications. The content that can be created within Rational Software Architect includes all kinds of UML 2.2 diagrams. It also includes features for automatic code generation starting from the models developed within the application.

All content created in Rational Software Architect can be published to HTML and deployed to Web servers for distributed viewing.

Rational Software Architect can also be connected to a number of other Rational lifecycle process tools in order to be fully used into the software process.

This tutorial is based on version 8.5.1 of RSA and explains the basic features to help get started with Rational Software Architect.

More information can be found in the IBM web site: http://www-01.ibm.com/software/rational/products/swarchitect/

If you have comments or questions regarding this document or Rational Software Architect, please contact **catedra.ibm.fiupm@gmail.com**

2. Terminology

UML: the Unified Modeling Language is a standardised general-purpose modelling language in the field of object-oriented software engineering. The Unified Modelling Language includes a set of graphic notation techniques to create visual models of object-oriented software-intensive systems.

GUI: Graphical User Interface.

Use Case Diagram: diagram used to describe a set of actions (use cases) that some system or systems (subject) should or can perform in collaboration with one or more external users of the system (actors).

Class Diagram: is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among the classes.

Sequence Diagram: is a kind of interaction diagram that shows how processes operate with one another and in what order.

Activity Diagram: are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency.

Communication Diagram: in UML 2.0, it is a simplified version of the UML 1 collaboration diagram. It models the interactions between objects or parts in terms of sequenced messages.

3. Application description

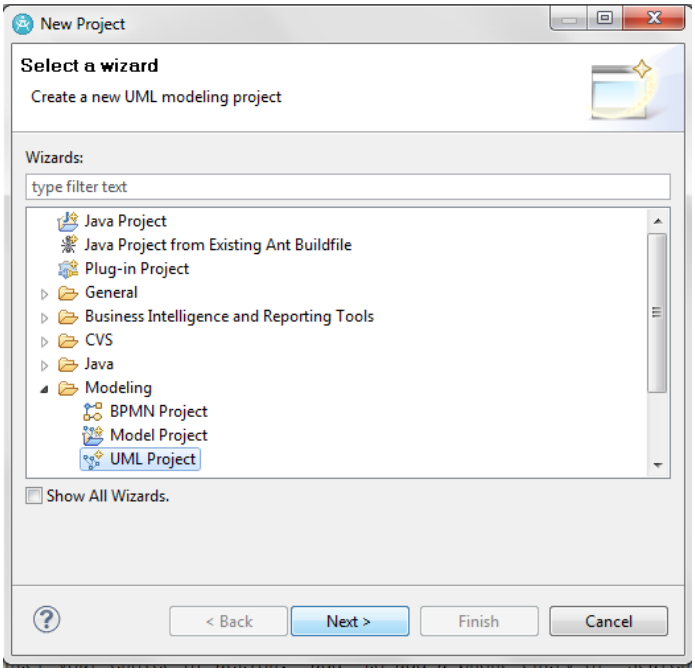
For this simple example we will model a phone book application. The user will be able to add entries that will be an association of a name to a phone number. And then the user will be able to search for a phone number by entering a name. For this example this application will not have persistent data.

The application will have 3 classes. A view will show the GUI of the application, a model will save the entries and search on the saved data, and a controller will react to the user inputs.

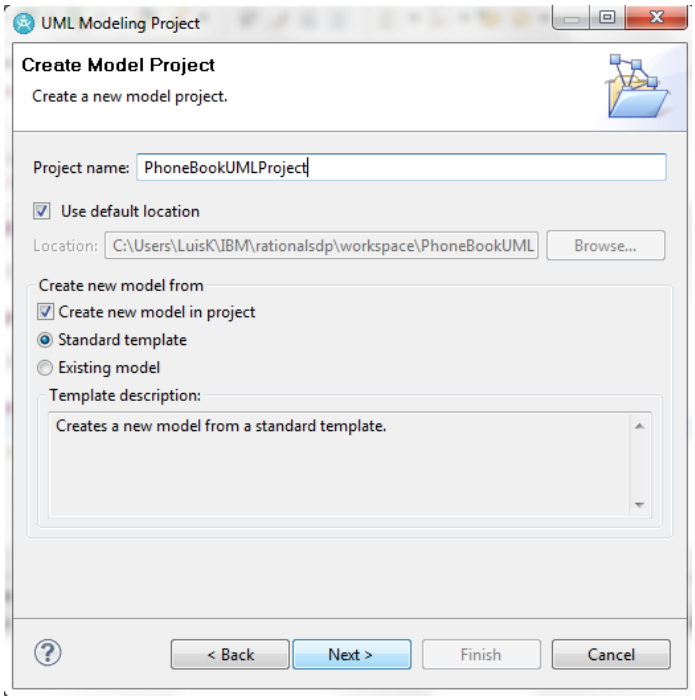
4. Creating the project

1. Go to **File** > **New** > **Project**.

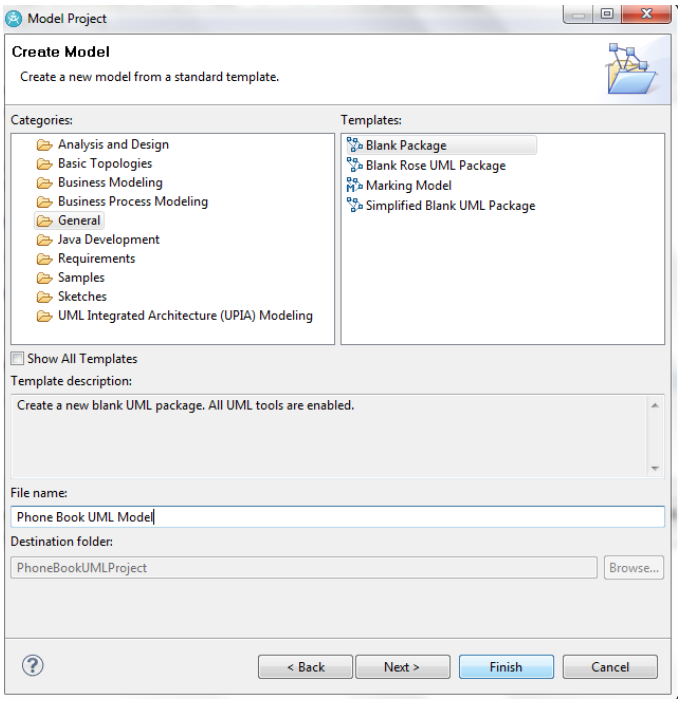
2. Select the **UML Project** wizard. Click **Next**



3. Enter a Project Name. Click **Next.**



4. On the Create Model Step, choose category **General** and template **Blank Package** and enter a model name. Click **Finish.**



5. Now you can see your UML Project with two subdirectories. The first subdirectory is Diagrams, which will only show the UML diagrams you create organized by the type of diagram. The second subdirectory, Models, will show the diagrams and all the UML objects that you create within the model.

5. Creating UML Diagrams

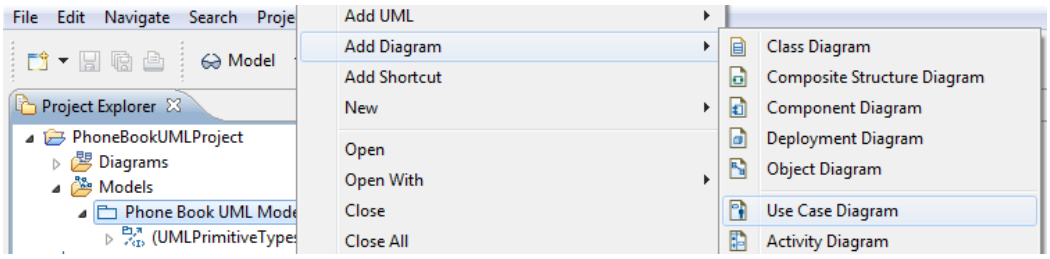
RSA includes features to create the following of UML diagrams:

- Class diagram
- Composite Structure Diagram
- Component Diagram
- Deployment Diagram
- Object Diagram
- Use Case Diagram
- Activity Diagram
- State Machine Diagram
- Timing Diagram
- Sequence Diagram
- Communication Diagram
- Interaction Overview Diagram
- Freeform Diagram: lets you create diagrams without the restrictions of any of the other diagram types.

In this tutorial we will create 5 diagrams, a use case diagram, a class diagram, a sequence diagram, a communication diagram and an activity diagram.

5.1 Use Case Diagram

1. Right click the **Phone Book UML Model** and select **Add Diagram > Use Case Diagram.**

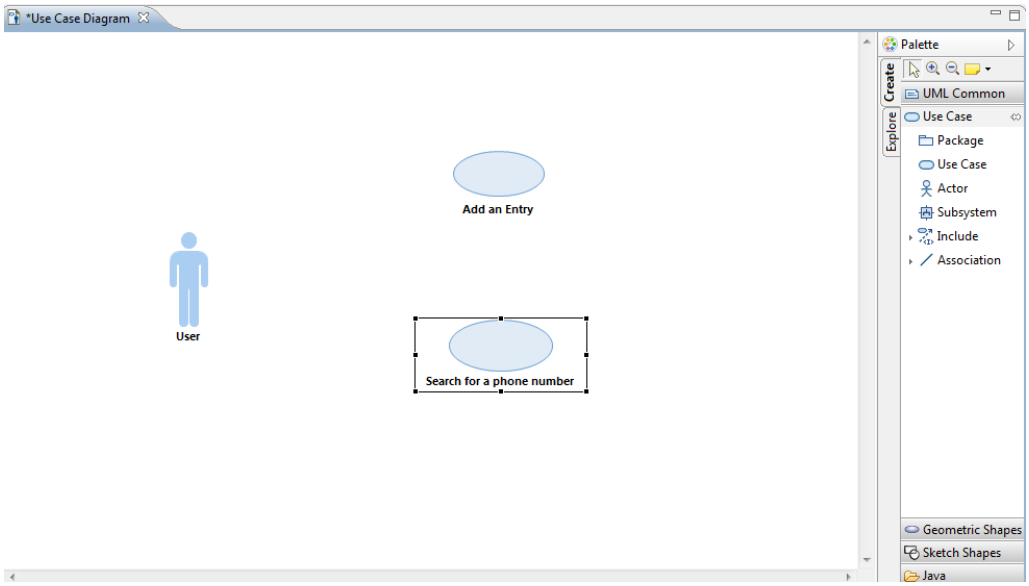


2. Enter a name for your diagram. Now you will see a pane where you will be able to add items from the palette to the diagram.

3. Click **Actor** in the palette and then click in the pane in order to add it. Name it “User”.

4. Click **Use Case** in the palette and then click in the pane in order to add it. Name it "Add an Entry"

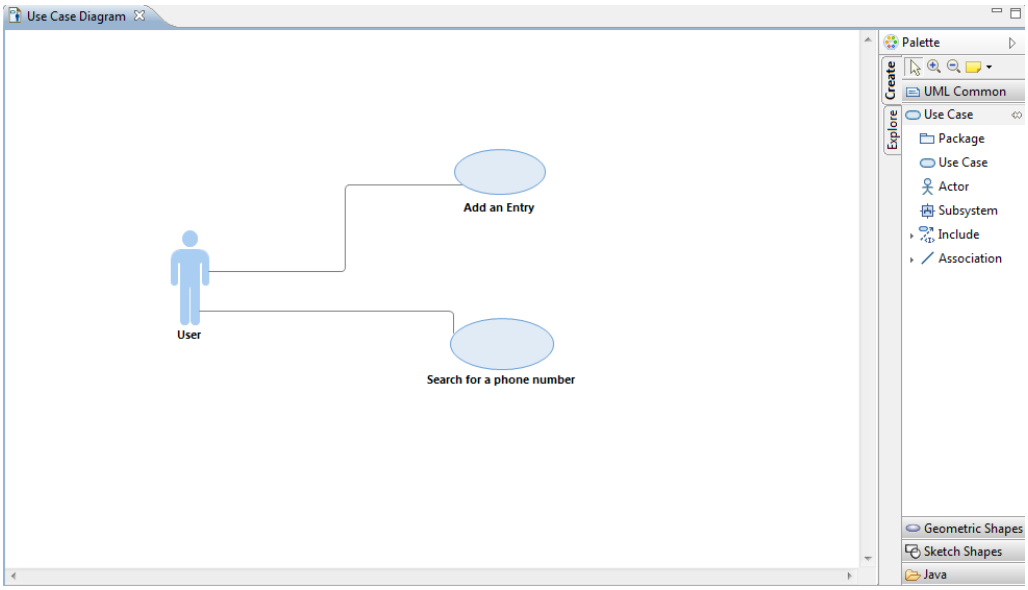
5. Click **Use Case** in the palette and then click in the pane in order to add it. Name it "Search for a phone number"



6. Click **Association** in the palette and then click and drag the association from "User" to "Add an Entry".

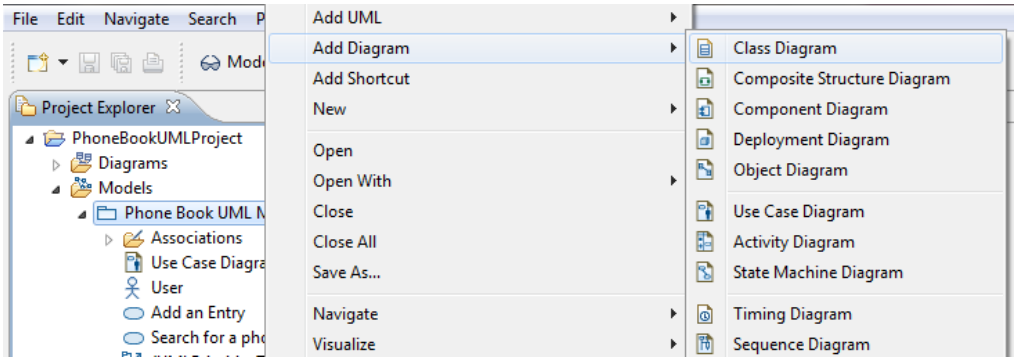
7. Click **Association** in the palette and then click and drag the association from "User" to "Search a phone number".

8. Now you have the complete use case diagram for the application. Save it.



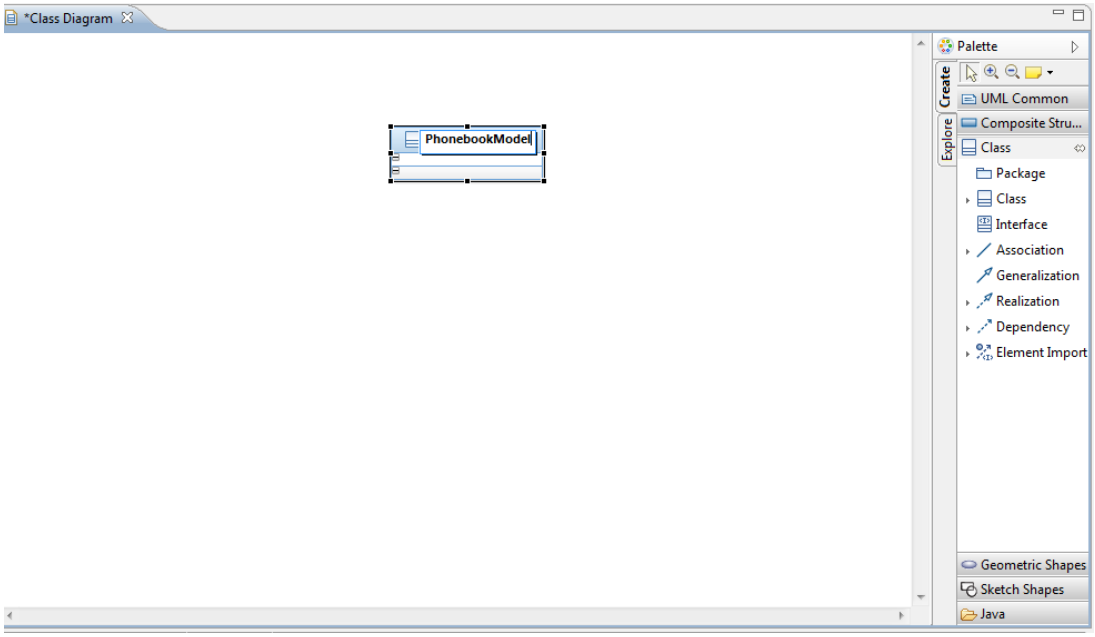
5.2 Class Diagram

1. Right click the **Phone Book UML Model** and select **Add Diagram > Class Diagram.**

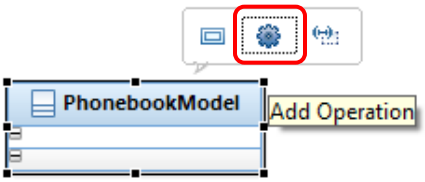


2. Enter a name for your diagram. Now you will see a pane where you will be able to add items from the palette to the diagram.

3. Click **Class** in the palette and then click in the pane in order to add it. Name it "PhonebookModel".



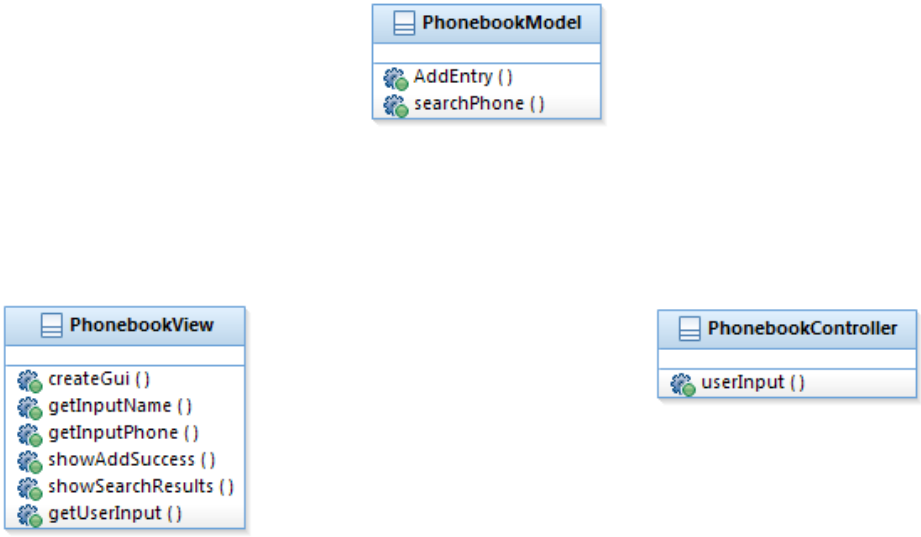
4. Hover with your mouse over the newly added class and click the add operation button that will appear.

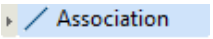


5. Name the operation “AddEntry”.

6. Repeat steps 3 to 5 with the following classes and operations:

Class	Operations
PhonebookModel	searchPhone
PhonebookController	userInput
PhonebookView	createGui getInputName getInputPhone showAddSuccess showSearchResults getUserInput

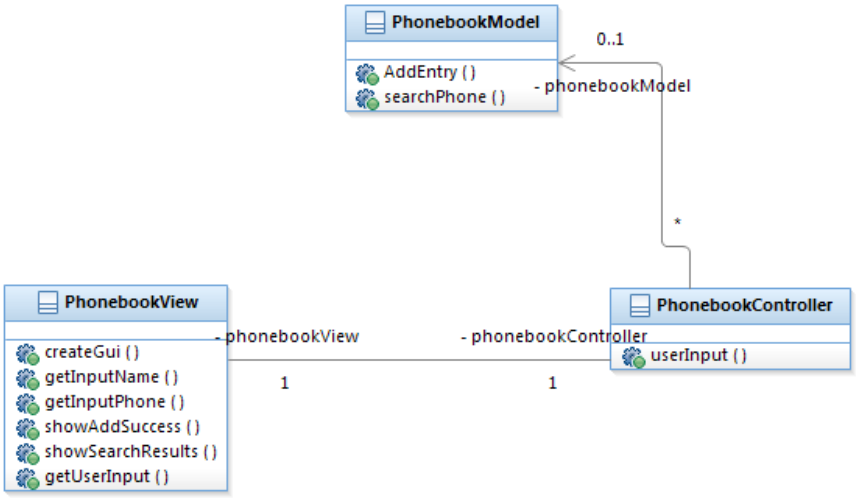


7. Now go to the palette and click the arrow next to the Association item 

in order to see the Association types. Click **Directed association** then click and drag the association from “PhonebookController” to “PhonebookModel”.

8. Repeat the process but this time adding a regular association between “PhonebookController” and “PhonebookView”.

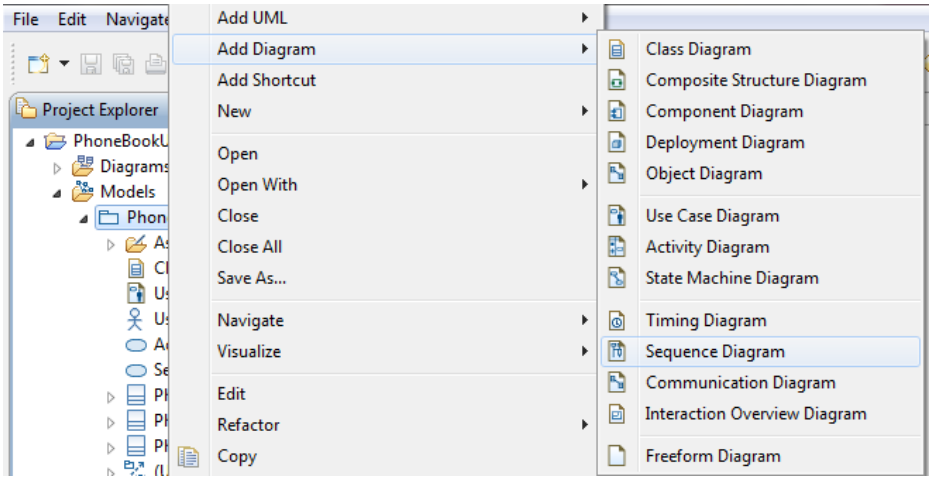
9. Save the final diagram.



5.3 Sequence Diagram

In this part of the tutorial we will create a sequence diagram for the use case when the user adds an entry to the phone book. If you want to create the diagram for the other use case, follow a similar approach.

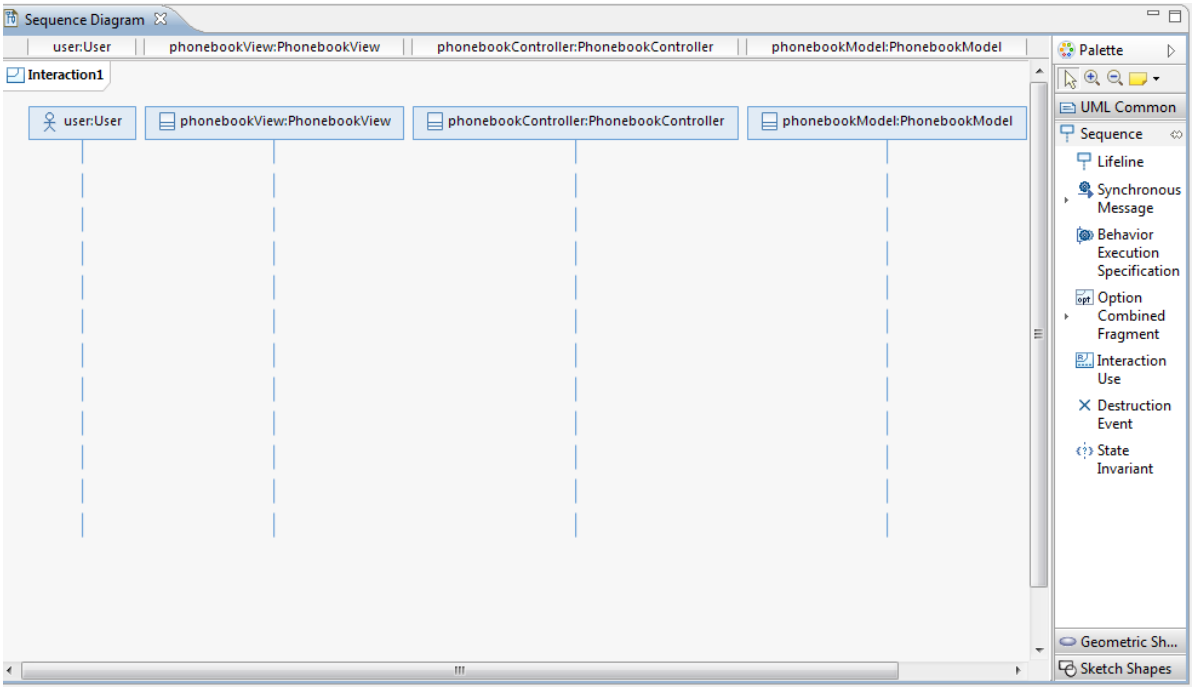
1. Right click the **Phone Book UML Model** and select **Add Diagram > Sequence Diagram.**



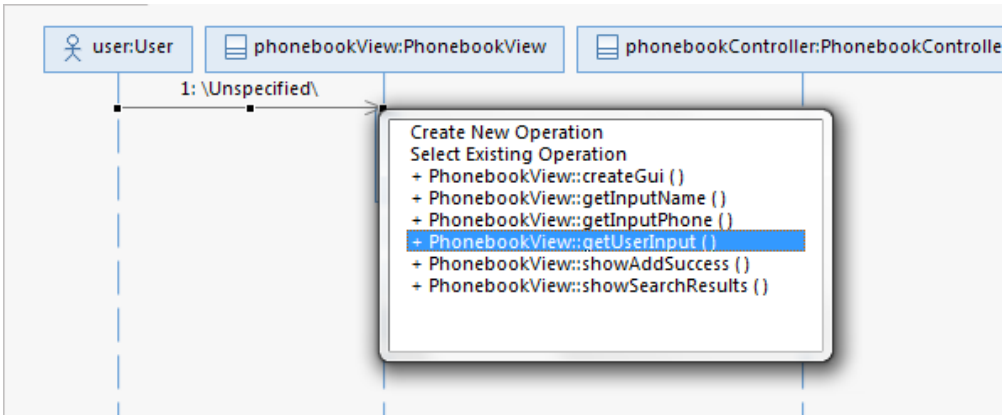
2. Enter a name for your diagram. Now you will see a pane where you will be able to add items from the palette and from the project explorer to the diagram. In order to have a better model traceability, which is recommended, it is better to grab the elements from the project explorer.

3. Drag the “User” from the project explorer to the diagram in order to add its lifeline.

4. Do the same with the classes “PhonebookView”, “PhonebookController” and “PhonebookModel”.



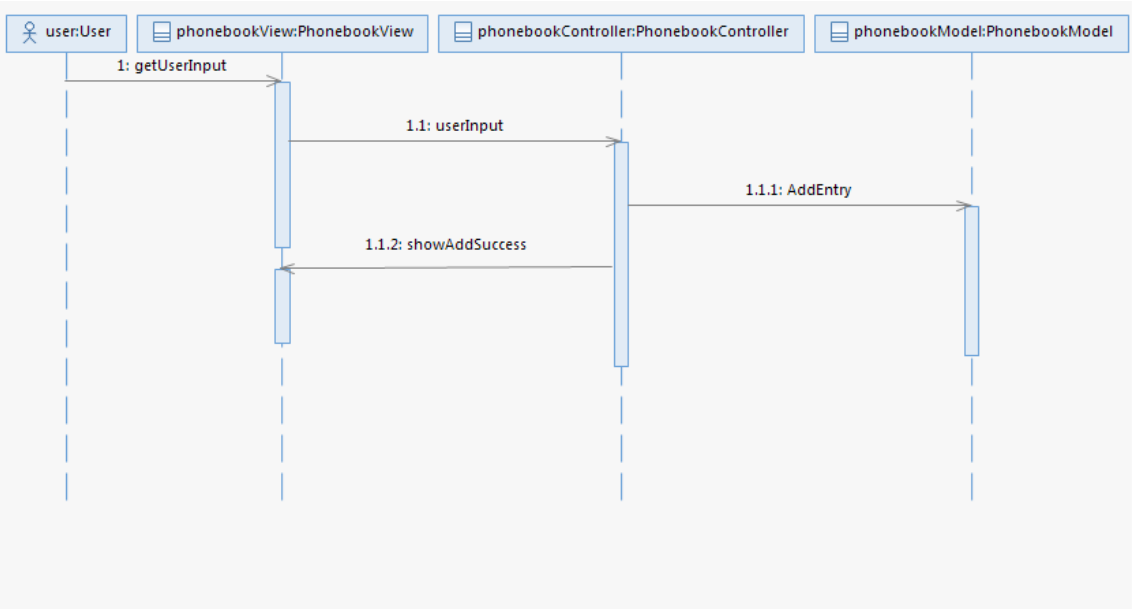
5. Select **Asynchronous Message** in the Palette. As shown in the next figure, click the line under **user: User** and then the line under **phoneBookView:PhoneBookView**. Select the operation **PhoneBookView::getUserInput()** from the drop down list.



6. Repeat step 5 with the following asynchronous messages:

From	To	Message
PhonebookView	PhonebookController	userInput
PhonebookController	PhonebookModel	addEntry
PhonebookController	PhonebookView	showAddSuccess

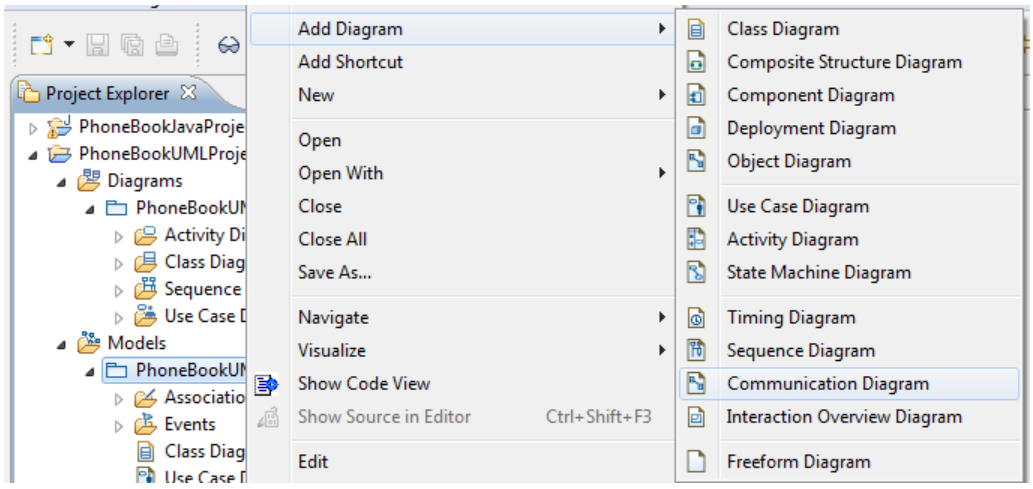
7. Save the final diagram.



5.4 Communication Diagram

In this part of the tutorial we will create a communication diagram related to the sequence diagram of the previous section.

1. Right click the **Phone Book UML Model** and select **Add Diagram > Communication Diagram.**

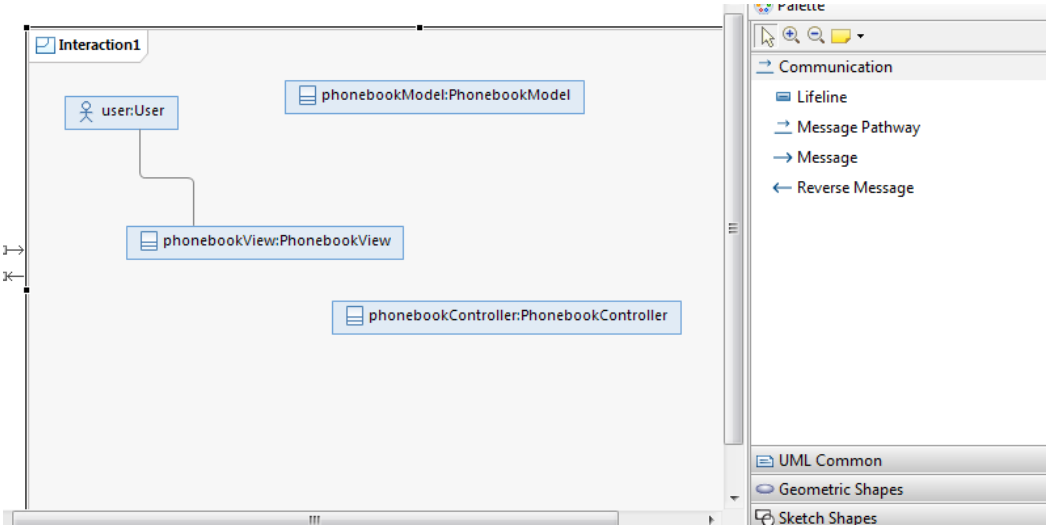


2. Enter a name for your diagram. Now you will see a pane where you will be able to add items from the palette and from the project explorer to the diagram. In order to have a better model traceability, which is recommended, it is better to grab the elements from the project explorer.

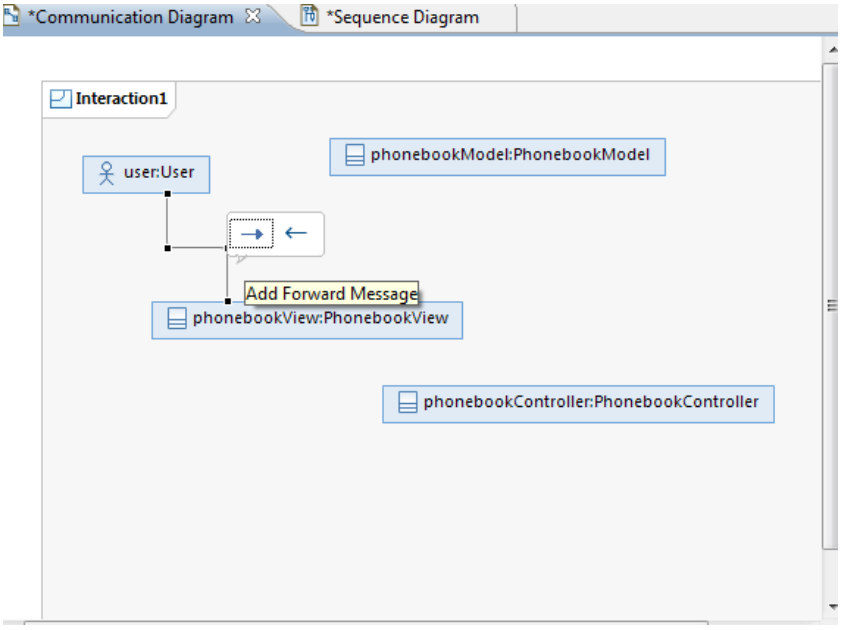
3. Drag the “User” from the project explorer to the diagram in order to add its lifeline.

4. Do the same with the classes “PhonebookView”, “PhonebookController” and “PhonebookModel”.

5. Select **Message Pathway** in the Palette. As shown in the next figure, click the line under **user: User** and then the line under **phoneBookView:PhoneBookView**.

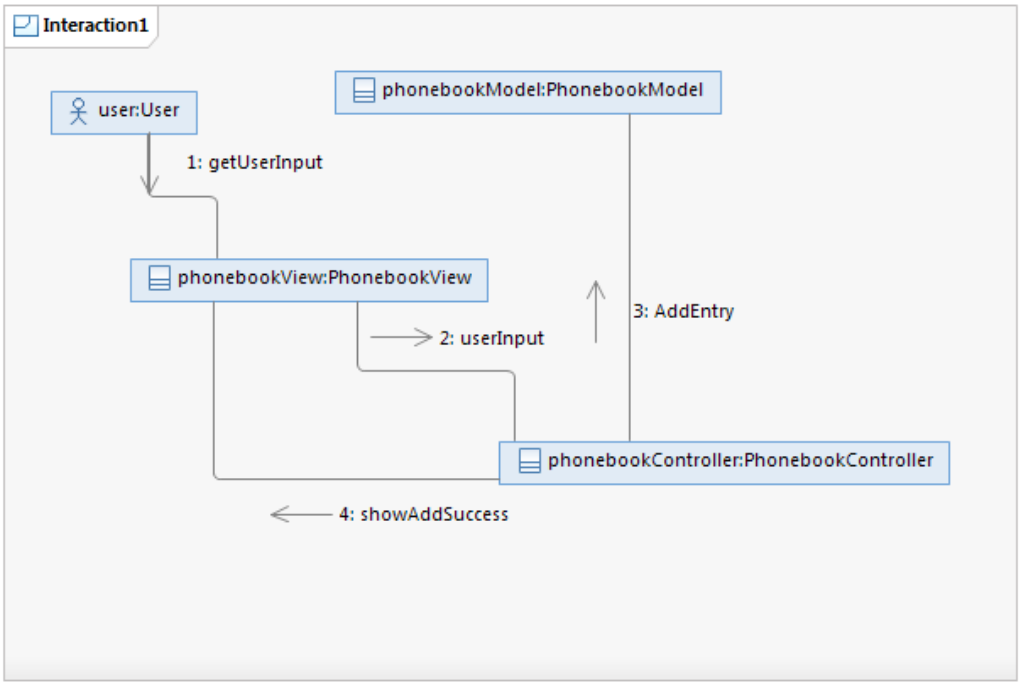


6. Hover over the newly added Message Pathway and click the **Forward Message icon.**



7. Select the operation **PhoneBookView::getUserInput()** from the drop down list. Erase the reverse message that goes from PhonebookView to User, which is automatically created.

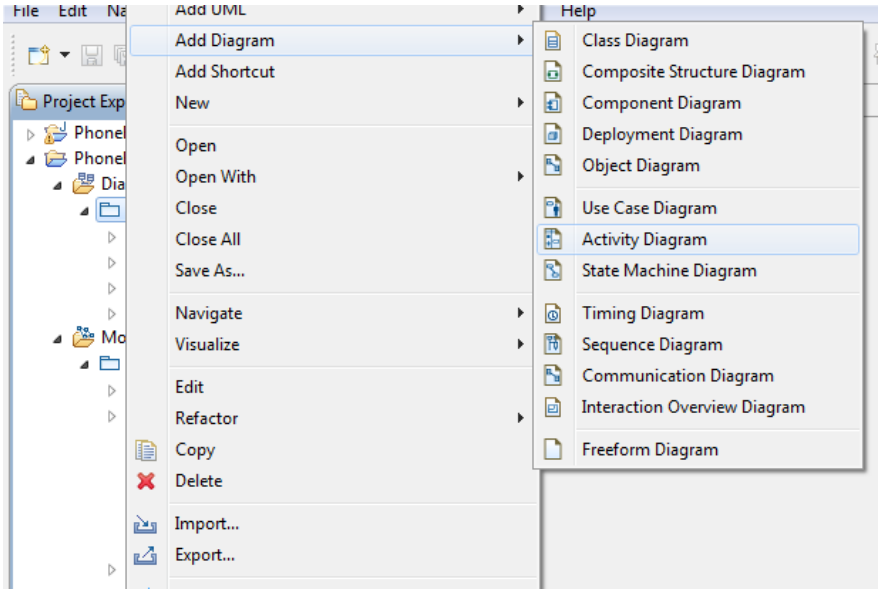
8. Repeat steps 6 and 7 in order to copy the operations added in the sequence diagram created in the previous section. You should end up with a diagram similar to the following figure.



5.5 Activity Diagram

In this part of the tutorial we will create an activity diagram for the application.

1. Right click the **Phone Book UML Model** and select **Add Diagram > Activity Diagram.**



2. Enter a name for your diagram. Now you will see a pane where you will be able to add items from the palette and from the project explorer to the diagram.

3. Select **Initial** from the palette and click in the pane to add it. Add the name “User runs application”

4. In the same way add the following **Actions:** “System waits for input”, “User enters name”, “User enters phone”, “User closes application”, “User clicks save”, “System saves entry”, “User clicks search”, “System shows search result”, “User closes application”.

5. Add an **Activity final** named “Application is closed”.

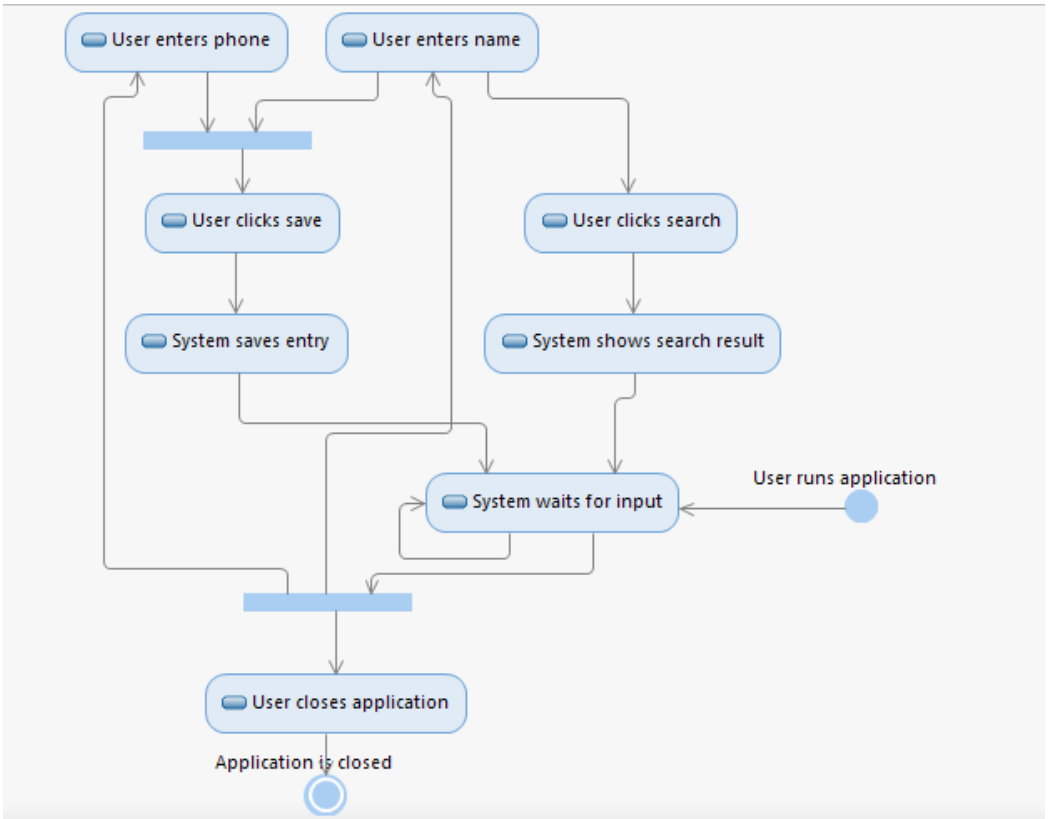
6. Add a fork and a join.

7. Add the following links:

From	To
User runs application	System waits for input
System waits for input	System waits for input
System waits for input	Fork
Fork	User enters name
Fork	User enters phone
Fork	User closes application
User enters name	Join
User enters name	User clicks search
User enters phone	Join
User closes application	Application is closed
Join	User clicks save
User clicks search	System shows search result
User clicks save	System saves entry
System shows search result	System waits for input
System saves entry	System waits for input

8. Right click the activity pane and click **Arrange all.** RSA will automatically arrange your activity diagram to make it easier to read.

9. You should end up with an activity diagram similar to the following:



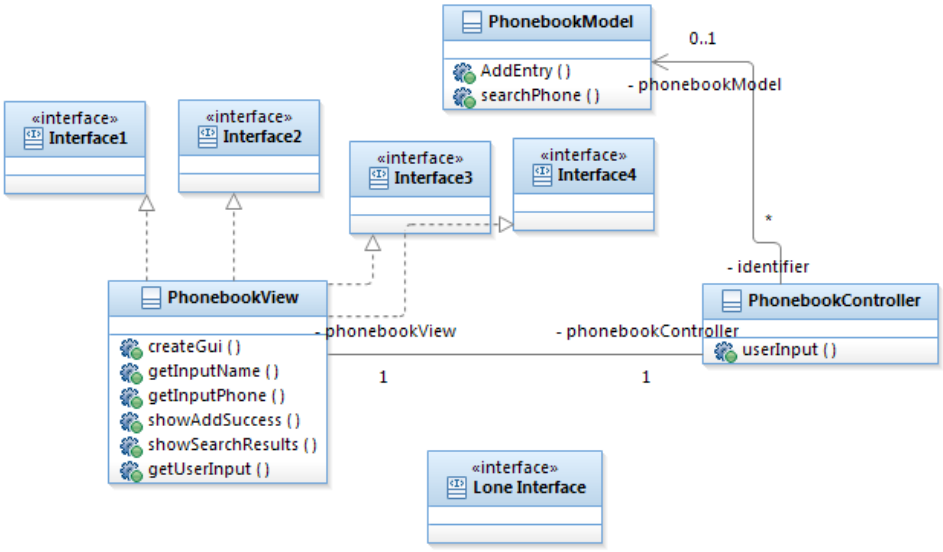
6. Analysing the design

RSA includes features analyse the quality of your models and implementation in a sub feature called Software Analyser. The analysis depends on a set of rules; the combination of rules will define the completeness and roughness of the analysis.

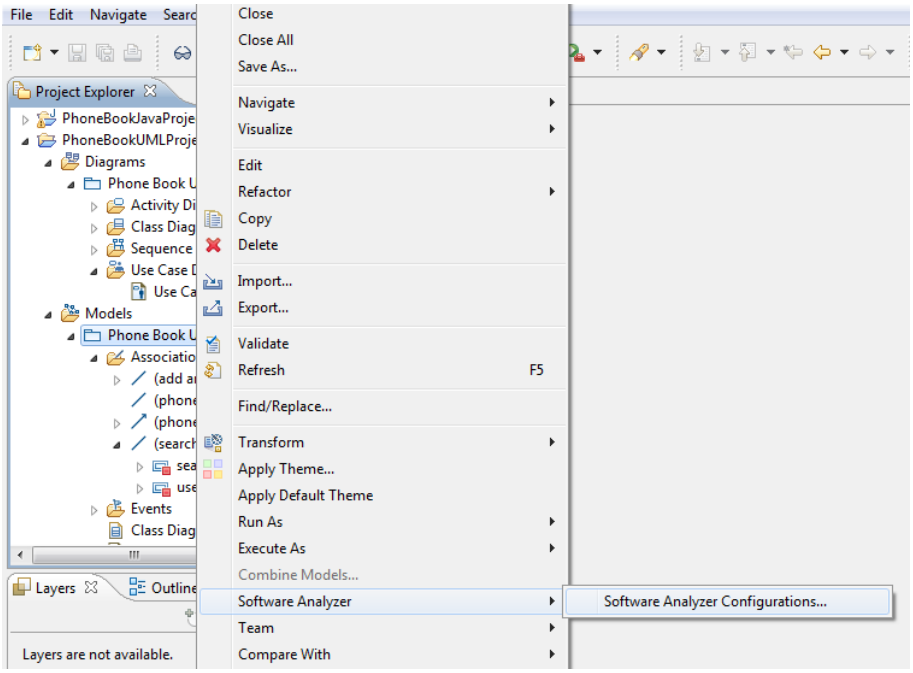
6.1 Creating the analysis configuration

1. In order to show the capabilities of the analyser add some errors or warnings in the model (this because the current model is too simple and is difficult to have important errors). Add the following:
    - Add an interface to the Class Diagram, open the class diagram, click **Interface** in the palette and add it to the pane. This will be an unimplemented abstract class, so the analyser will give you a warning for it.
    - Add 4 more interfaces to the Class Diagram and make one of your classes implement all of them (by adding an **Interface Realization** relationship). This will simulate a class that is getting too big and "knowing it all", so a warning from the analyser will tell you to take a look at that class and if you really need to implement all that interfaces.

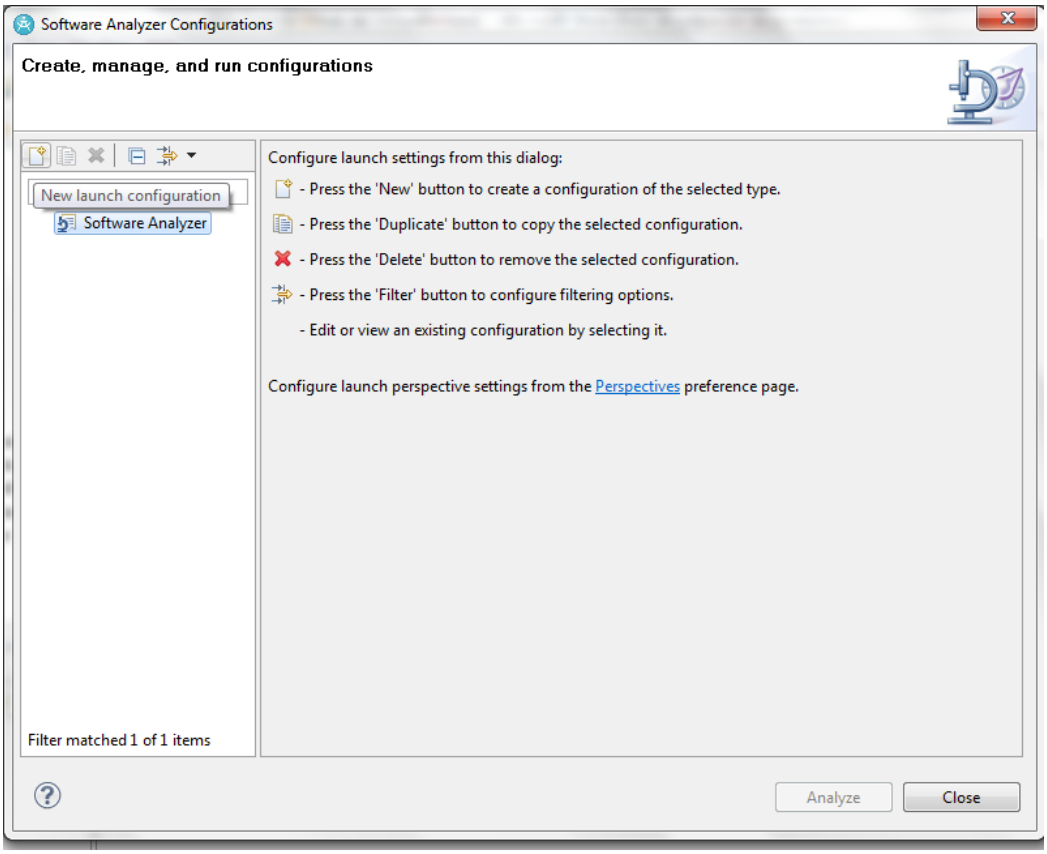
You will end up with a class diagram similar to the following:



2. Right click the **Phone Book UML Model** and select **Software Analyzer > Software Analyzer Configurations.**

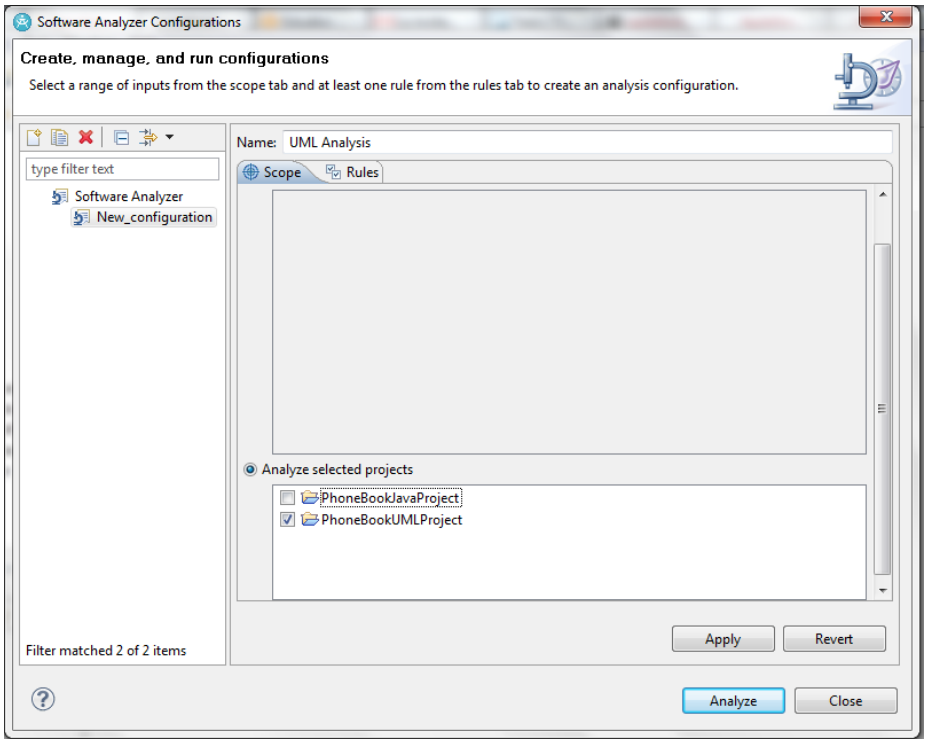


3. On the Software Analyzer Configurations window, click **Software Analyzer** and click the **New button** to add a new configuration.

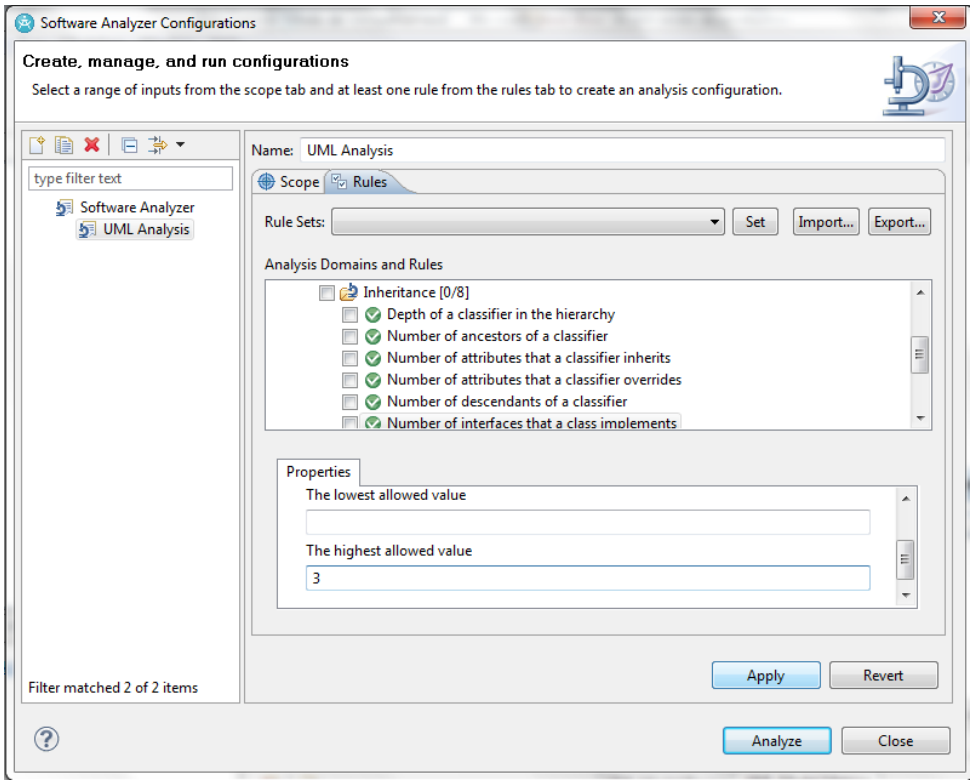


4. Name it **UML Analysis.**

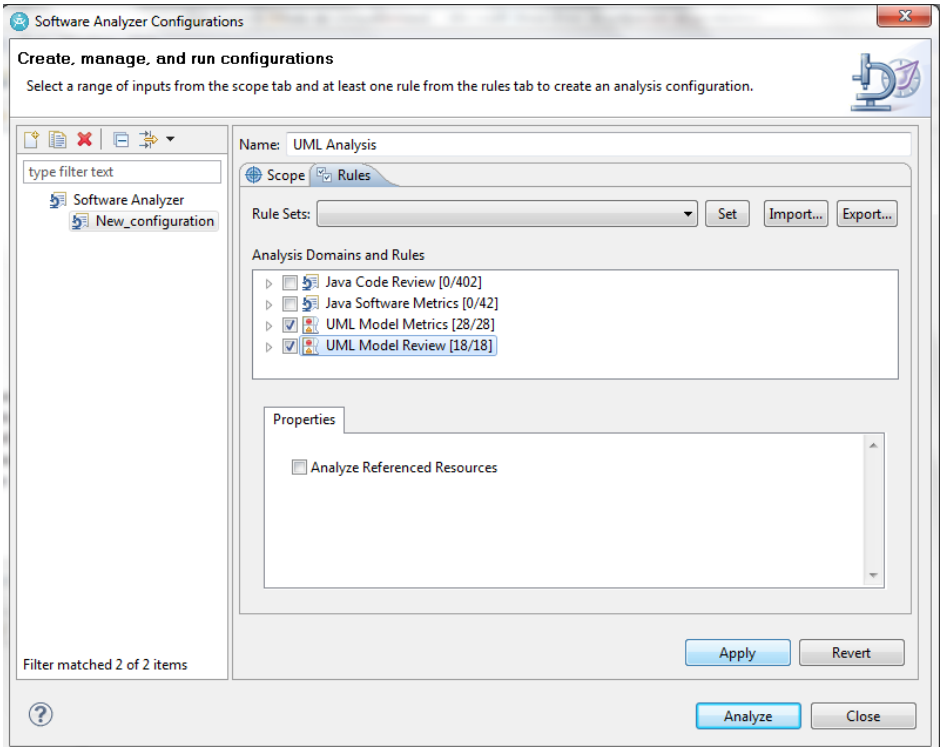
5. On the scope tab select the **Analyze selected project radio button** and the check the **PhonebookUMLProject checkbox.**



6. On the Rules tab you can see all the available rules that can be selected for the analysis, by expanding the Rule trees you can get into the rules and modify them. For example you can change the severity of the rule, the accepted minimum value and the accepted maximum value. For this example, look for the rule under **UML Model Metrics > Inheritance > Number of interfaces that a class implements** set the maximum value to 3 and click **Apply** (note that this is just an example, 3 or even more is a fairly accepted number of implemented interfaces for a class).



7. For this analysis check the **UML Model Metrics checkbox** and the **UML Model Review checkbox** in order to use all the available UML analysis rules (Some of them will have the minimum and maximum accepted values empty, so if you want real feedback you have to fill the accepted values as explained in the previous step).

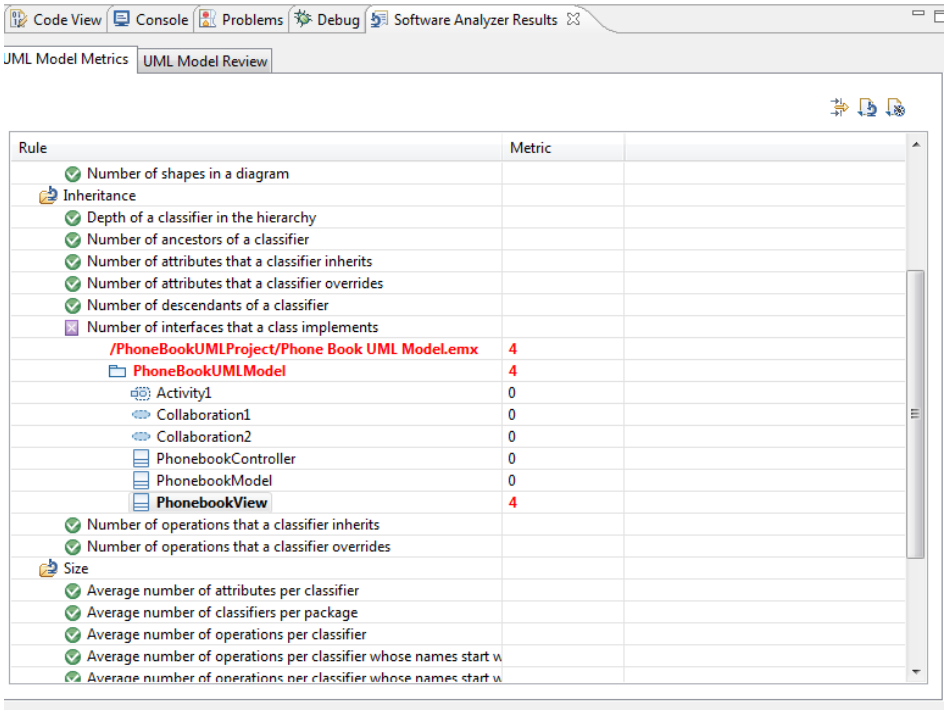


8. Click **Apply** and the **Analyze.**

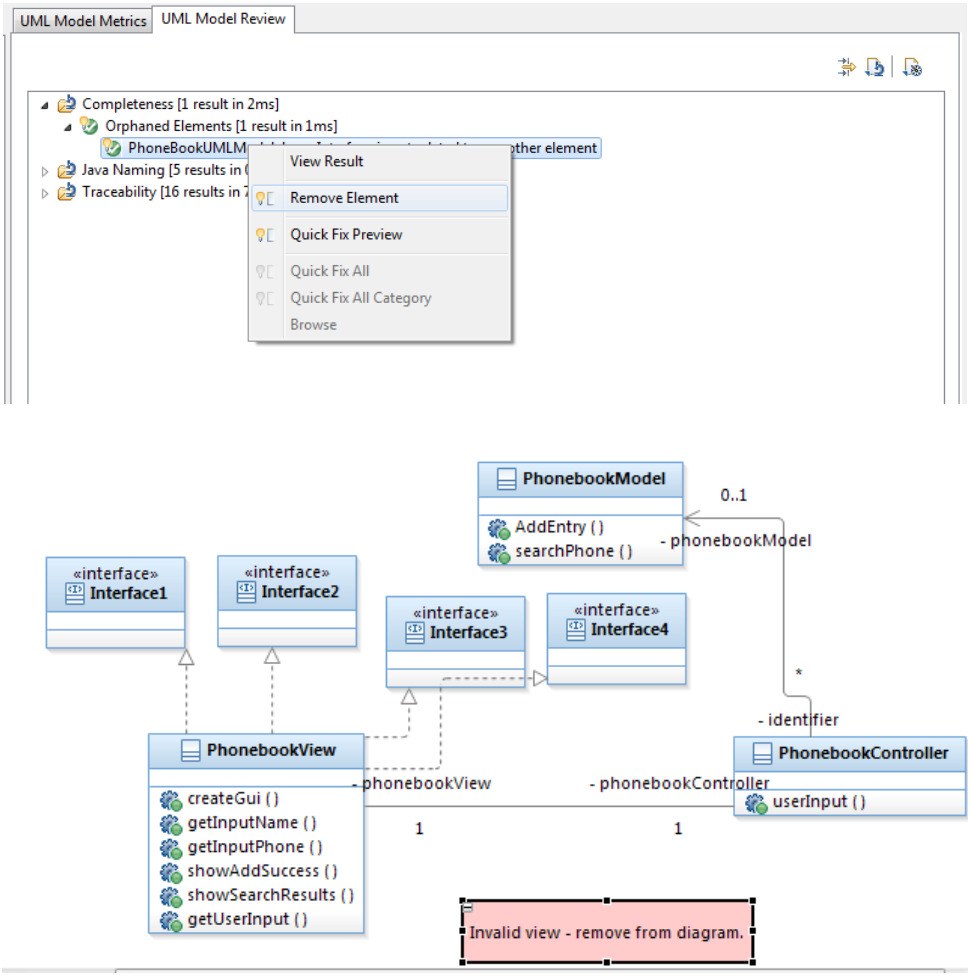
6.2 Viewing and reporting the analysis results

1. On the Software Analyzer Results view you will be able to see the results of the analysis off your UML models.

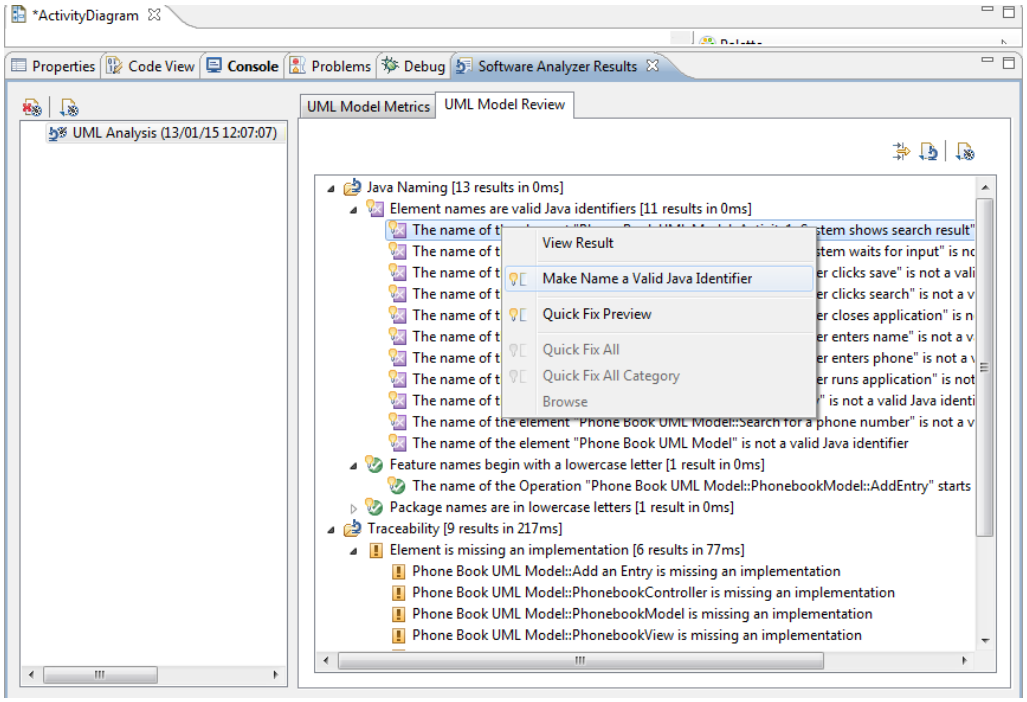
2. On the UML Model Metrics tab you can expand the rules trees in order to see the metrics of each part of the model related to the metric you are viewing. If you open the Inheritance section, you will see the warning on **Number of interfaces that a class implements**, you can further expand the tree and see which is the problematic class.



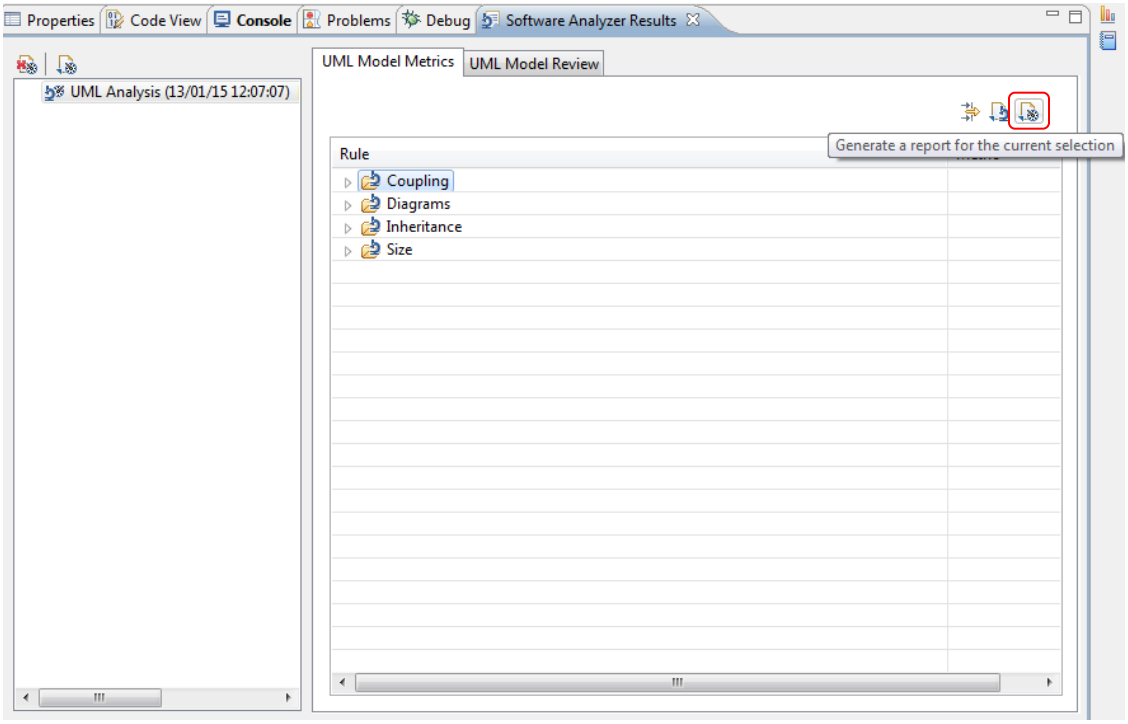
3. Go to the UML Model Review Tab. Open the **Correctness section** and you will see the Orphaned interface that we left empty and with no relations to the other classes. RSA will give a quick fix to this such as removing the element. Right click the leaf of the warning and click on **Remove Element** in order to remove this unused element from the model. Then if you open the class diagram it will show you the place on which this orphaned element was before as an Invalid View to give some warning in case you miss removed it. Click on the element and remove it also from the class diagram.



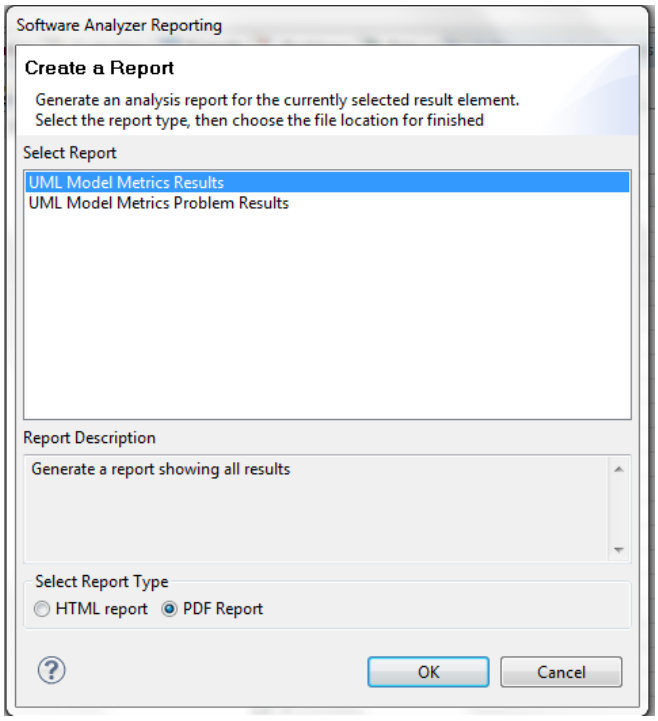
4. In the same UML Model Review Tab, you can fix other parts of the models such as naming standards in the Java Naming section. In order to fix them right-click on one of them and click **Make name Valid Java Identifier.** This will change the name of the artifact in order to make it a valid java name (Usually removing the spaces). Correct all the severe problems.



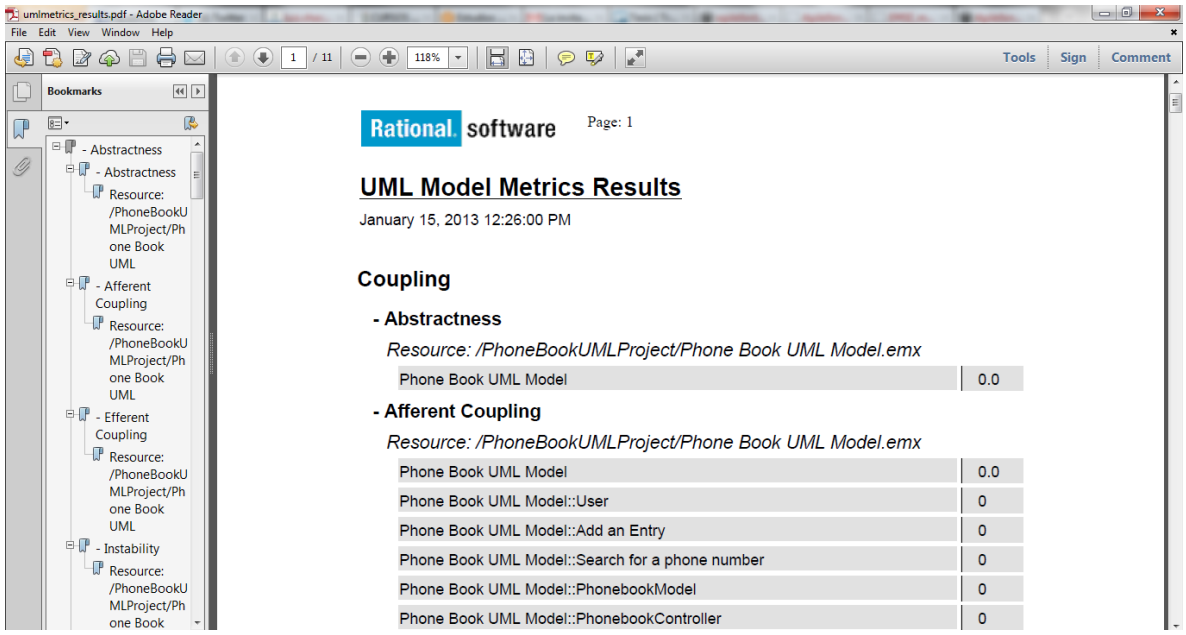
5. From the Software Analyzer you can also create reports. Go to either of the tabs and click the **generate report button.**



6. On the Software Analyzing Reporting window, select your Report and your Report type and click **OK.**



7. RSA will generate a report which you can browse and share with other stakeholders of the project.



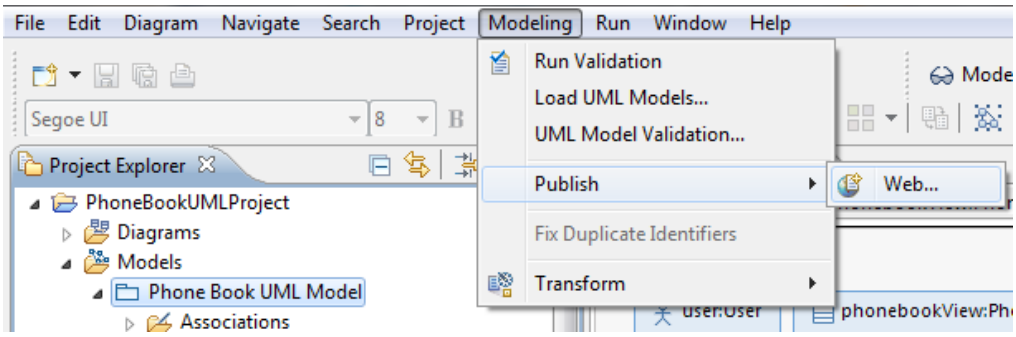
8. Finally go back to your class diagram and remove the added interfaces, also remove them from the model directory.

7. Publishing the design

RSA includes features to publish your design in order to share it with other stakeholders. Your design will be published in HTML format.

1. Click the **Phone Book UML Model.**

2. Go to **Modeling** > **Publish** > **Web**.



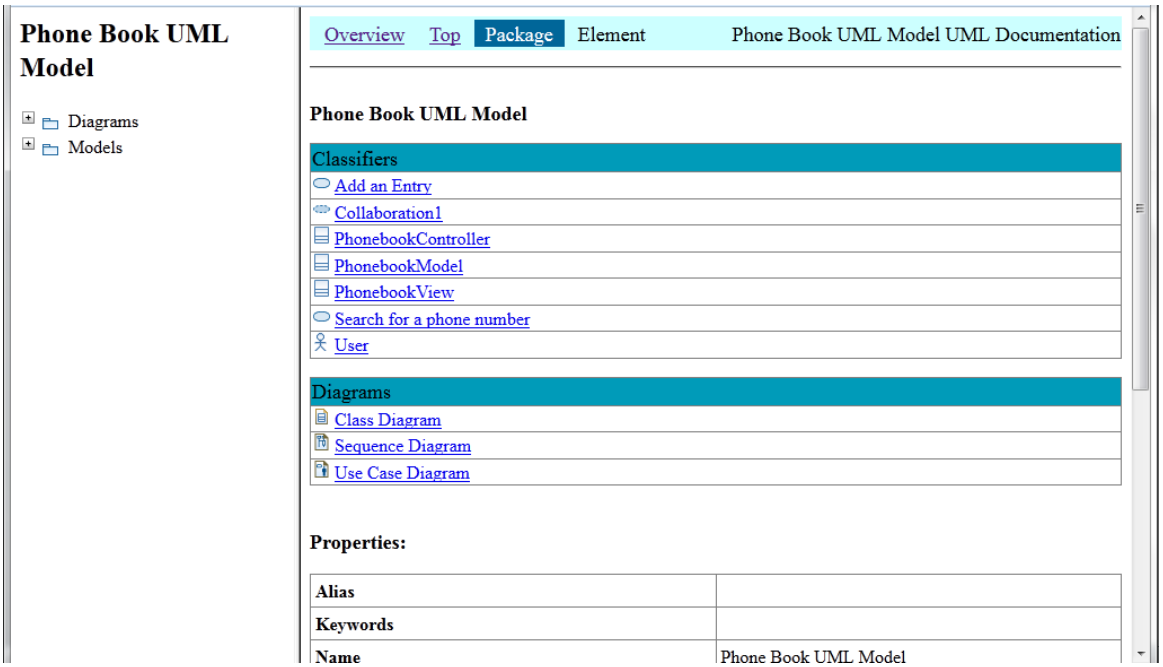
3. Enter the desired folder where you want to publish your design.

4. Go to the selected folder and open the index.html.

Published Items

Phone Book UML Model

5. Click the **Phonebook Model link** and navigate through your design.



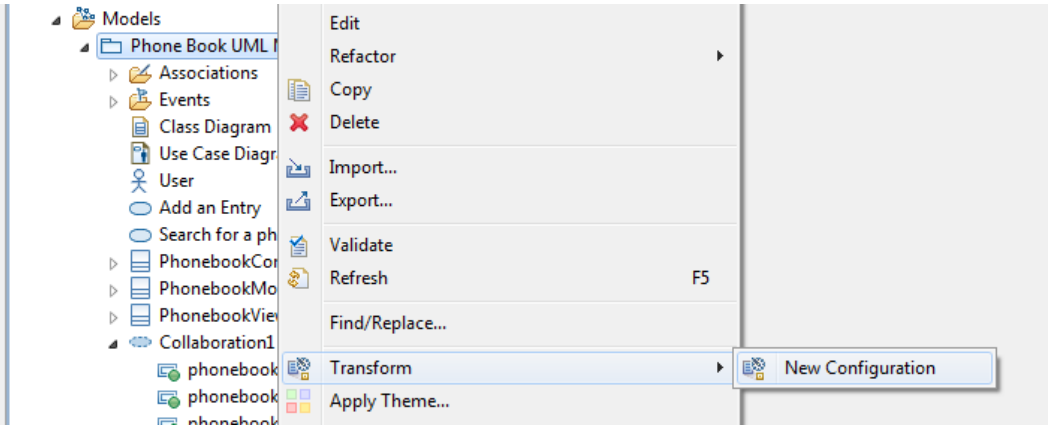
8. Transforming the UML design into code

RSA includes features to transform your design into a number of programming languages, so you begin your implementation phase with a skeleton of the program and fill it with all the implementation details. In this tutorial we will generate Java code from our design. Other available transformations in RSA are the following:

- UML object model to SQL logical data model (IBM InfoSphere Data Architect)
- UML to XSD; XSD to UML
- UML to Java, Java to UML
- UML to JPA, JPA to UML
- UML to C#, C# to UML
- UML to VB.NET, VB.NET to UML
- UML to CORBA

8.1 Creating the Java project

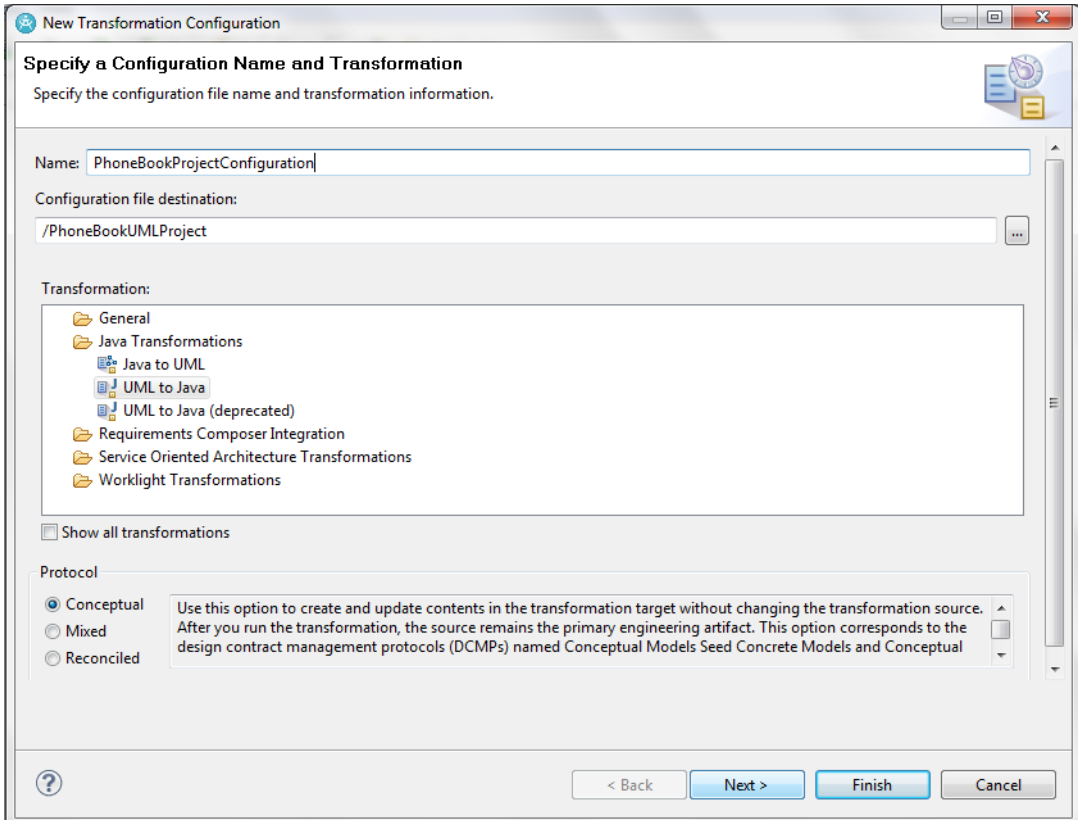
1. Right click the **Phone Book UML Model** and select **Transform > New Configuration.**



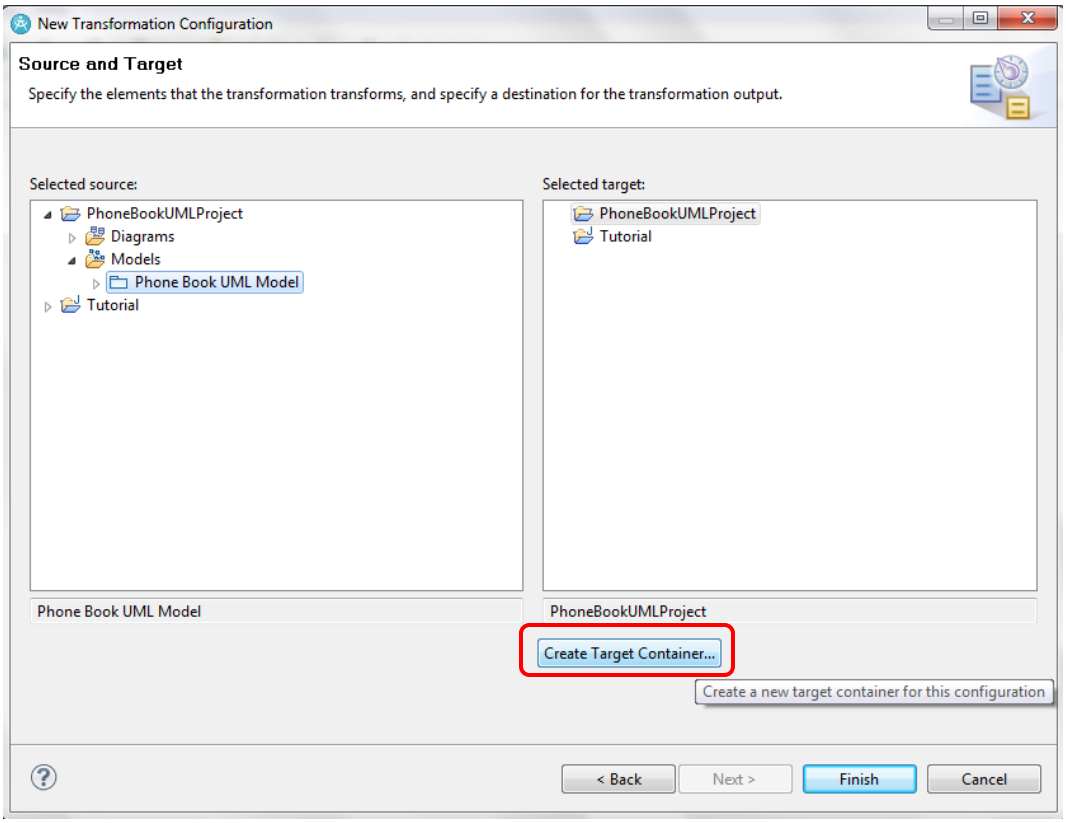
2. Enter the name of the configuration.

3. On the Transformations list select **Java Transformations > UML to Java.**

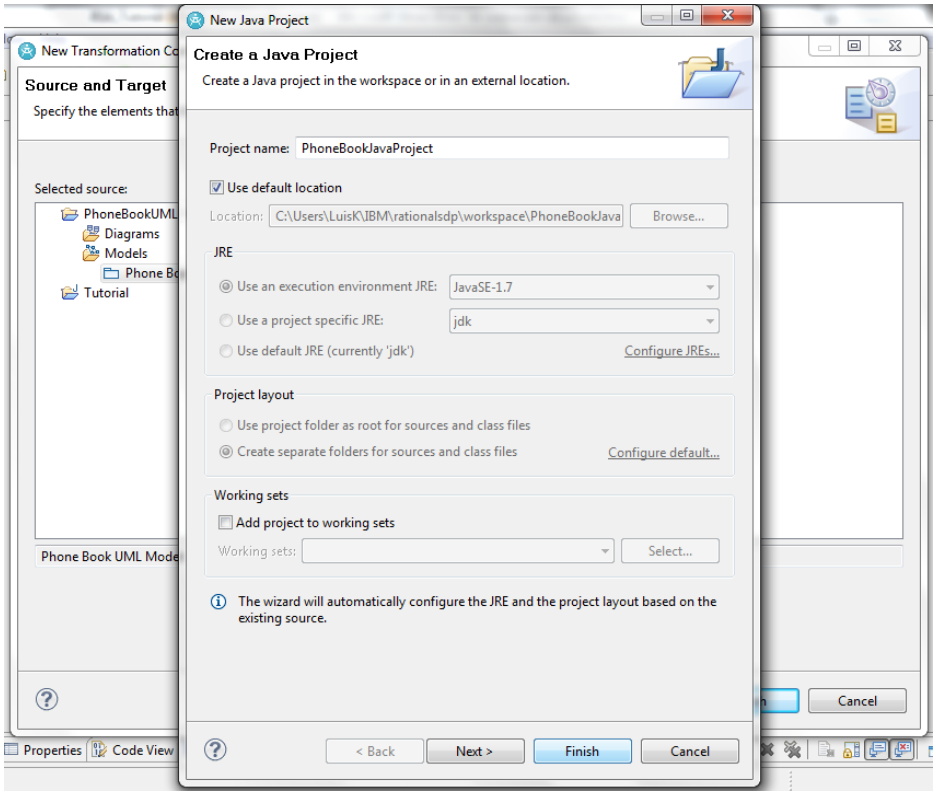
4. Click **Next.**



5. Now select your target, for doing this click the button **Create Target Container**



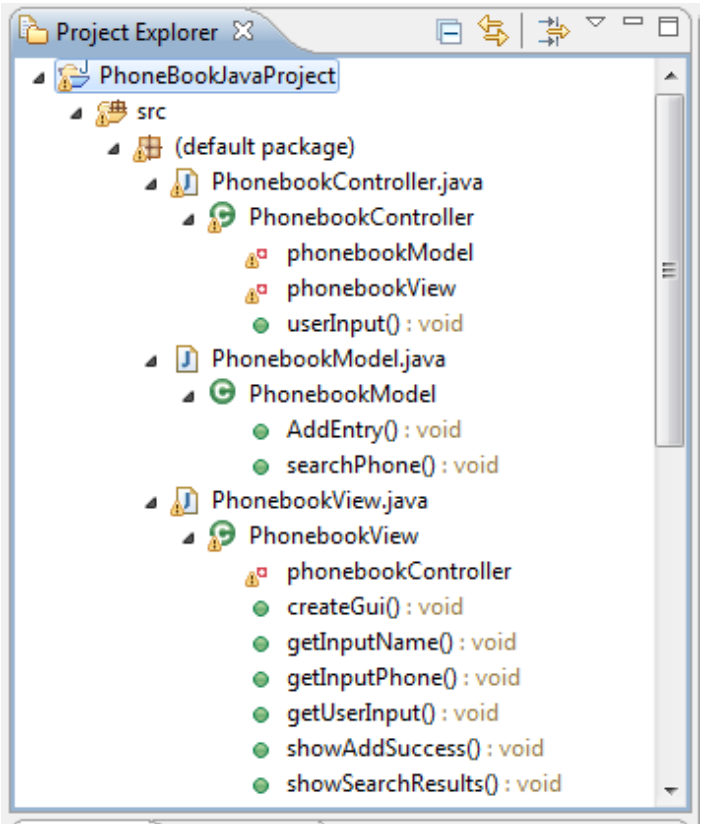
6. Now enter your project name and click **Finish.**



7. Click **Finish.**

8. Now you will see your new Java project created in your workspace. In order to fill it with the generated classes, open your newly created configuration file that must be called something like “PhonebookProjectConfiguration.tc” (it should have open automatically after the creation) and click **Run.**

9. Now you can go to your Java Project and see the 3 classes and its methods as you wrote them in the class diagram.



8.2 Implementation phase

This tutorial assumes you have a basic knowledge on Java programing so the implementation details will be overlooked.

1. Fill your classes in order to be similar as the following:

PhonebookView.java

```
import java.awt.Button;
import java.awt.FlowLayout;
import java.awt.Frame;
import java.awt.Label;
import java.awt.TextField;
```

```
  import java.awt.event.ActionEvent;
  import java.awt.event.ActionListener;
  import java.awt.event.WindowAdapter;
  import java.awt.event.WindowEvent;

  import javax.swing.JOptionPane;

  /**
   *
   */

  /**
   * <!-- begin-UML-doc -->
   * <!-- end-UML-doc -->
   * @author CatedraIBM
   * @generated "UML to Java
(com.ibm.xtools.transform.uml2.java5.internal.UML2JavaTransform)"
   */
  public class PhonebookView implements ActionListener {

       private Frame f;
       private TextField tName;
       private TextField tPhone;
       private Button bAdd;
       private Button bSearch;

       /**
        * <!-- begin-UML-doc -->
        * <!-- end-UML-doc -->
        * @generated "UML to Java
(com.ibm.xtools.transform.uml2.java5.internal.UML2JavaTransform)"
        */
       private PhonebookController phonebookController;

       /**
        * <!-- begin-UML-doc -->
        * <!-- end-UML-doc -->
        * @generated "UML to Java
(com.ibm.xtools.transform.uml2.java5.internal.UML2JavaTransform)"
        */
       public void createGui() {
              // begin-user-code
              f = new Frame();
              f.setTitle("Phonebook - Add entry or search for phone");
              f.setSize(900,70); // default size is 0,0
              f.setLocation(10,200); // default is 0,0 (top left corner)

              // Window Listeners
              f.addWindowListener(new WindowAdapter() {
                     public void windowClosing(WindowEvent e) {
                        System.exit(0);
                     } //windowClosing
              } );

              f.setLayout(new FlowLayout());

              Label desc = new Label("*For search only the \"Name\" textfield will
be used");
              Label lName = new Label("Name:");
              tName = new TextField(20);
              Label lPhone = new Label("Phone:");
              tPhone = new TextField(20);
              bAdd = new Button("Add Entry");
```

```
			bAdd.setActionCommand("add");
			bAdd.addActionListener(this);
			bSearch = new Button("Search");
			bSearch.setActionCommand("search");
			bSearch.addActionListener(this);
			f.add(desc);
			f.add(lName);
			f.add(tName);
			f.add(lPhone);
			f.add(tPhone);
			f.add(bAdd);
			f.add(bSearch);

			f.setVisible(true);
			// end-user-code
		}

		/**
		 * <!-- begin-UML-doc -->
		 * <!-- end-UML-doc -->
		 * @return
		 * @generated "UML to Java
(com.ibm.xtools.transform.uml2.java5.internal.UML2JavaTransform)"
		 */
		public String getInputName() {
			// begin-user-code
			return tName.getText();
			// end-user-code
		}

		/**
		 * <!-- begin-UML-doc -->
		 * <!-- end-UML-doc -->
		 * @generated "UML to Java
(com.ibm.xtools.transform.uml2.java5.internal.UML2JavaTransform)"
		 */
		public String getInputPhone() {
			// begin-user-code
			return tPhone.getText();
			// end-user-code
		}

		/**
		 * <!-- begin-UML-doc -->
		 * <!-- end-UML-doc -->
		 * @generated "UML to Java
(com.ibm.xtools.transform.uml2.java5.internal.UML2JavaTransform)"
		 */
		public void showAddSuccess() {
			// begin-user-code
			JOptionPane.showMessageDialog(null, "Added entry with Name = " +
getInputName() + " and Phone = " + getInputPhone());
			// end-user-code
		}

		/**
		 * <!-- begin-UML-doc -->
		 * <!-- end-UML-doc -->
		 * @generated "UML to Java
(com.ibm.xtools.transform.uml2.java5.internal.UML2JavaTransform)"
		 */
		public void showSearchResults(String phone) {
			// begin-user-code
```

```
            JOptionPane.showMessageDialog(null, "Phone for person with Name = "
+ getInputName() + " is " + phone);
            // end-user-code
      }

      /**
       * <!-- begin-UML-doc -->
       * <!-- end-UML-doc -->
       * @generated "UML to Java
(com.ibm.xtools.transform.uml2.java5.internal.UML2JavaTransform)"
       */
      public void getUserInput(String command) {
            // begin-user-code
            phonebookController.userInput(command);
            // end-user-code
      }

      public PhonebookView(PhonebookController pController) {
            phonebookController = pController;
      }

      @Override
      public void actionPerformed(ActionEvent e) {
            getUserInput(e.getActionCommand());
      }
  }
```

PhonebookModel.java

```
  import java.util.Hashtable;

  /**
   *
   */

  /**
   * <!-- begin-UML-doc -->
   * <!-- end-UML-doc -->
   * @author CatedraIBM
   * @generated "UML to Java
(com.ibm.xtools.transform.uml2.java5.internal.UML2JavaTransform)"
   */
  public class PhonebookModel {

      private Hashtable<String, String> table;

      /**
       * <!-- begin-UML-doc -->
       * <!-- end-UML-doc -->
       * @generated "UML to Java
(com.ibm.xtools.transform.uml2.java5.internal.UML2JavaTransform)"
       */
      public void AddEntry(String name, String phone) {
            // begin-user-code
            table.put(name,phone);
            // end-user-code
      }

      /**
       * <!-- begin-UML-doc -->
       * <!-- end-UML-doc -->
```

```
	 * @generated "UML to Java
(com.ibm.xtools.transform.uml2.java5.internal.UML2JavaTransform)"
	 */
	public String searchPhone(String name) {
		// begin-user-code
		return table.get(name);
		// end-user-code
	}

	public PhonebookModel() {
		table = new Hashtable<String, String>();
	}
  }
```

PhonebookController.java

```
  /**
   *
   */

  /**
   * <!-- begin-UML-doc -->
   * <!-- end-UML-doc -->
   * @author CatedraIBM
   * @generated "UML to Java
(com.ibm.xtools.transform.uml2.java5.internal.UML2JavaTransform)"
   */
  public class PhonebookController {
	/**
	 * <!-- begin-UML-doc -->
	 * <!-- end-UML-doc -->
	 * @generated "UML to Java
(com.ibm.xtools.transform.uml2.java5.internal.UML2JavaTransform)"
	 */
	private PhonebookModel phonebookModel;
	/**
	 * <!-- begin-UML-doc -->
	 * <!-- end-UML-doc -->
	 * @generated "UML to Java
(com.ibm.xtools.transform.uml2.java5.internal.UML2JavaTransform)"
	 */
	private PhonebookView phonebookView;

	/**
	 * <!-- begin-UML-doc -->
	 * <!-- end-UML-doc -->
	 * @generated "UML to Java
(com.ibm.xtools.transform.uml2.java5.internal.UML2JavaTransform)"
	 */
	public void userInput(String command) {
		// begin-user-code
		if(command.equals("add")) {
			phonebookModel.AddEntry(phonebookView.getInputName(),
phonebookView.getInputPhone());
			phonebookView.showAddSuccess();
		} else if(command.equals("search")) {

	phonebookView.showSearchResults(phonebookModel.searchPhone(phonebookView.ge
tInputName()));
		}
		// end-user-code
	}
```

```
    public PhonebookController() {
           phonebookModel = new PhonebookModel();
    }

    public void setView(PhonebookView pView) {
           phonebookView = pView;
    }

    public static void main(String[] args)
    {
           PhonebookController controller = new PhonebookController();
           PhonebookView view = new PhonebookView(controller);
           controller.setView(view);
           view.createGui();
    }
}
```

2. Run the sample project.

9. Other features

9.1 Integration with other Rational Tools

RSA includes features to connect to other Rational tools, this way it can be part of the fool software development lifecycle. The following are integrations it can do and a short description of their purpose:

- IBM® InfoSphere® Data Architect: Import and export data models.
- IBM® Rational® Requisite Pro: Import requirements to model.
- IBM® Rational® Asset Manager: Manage work products.
- IBM® Rational® Team Concert: Sharing projects, managing changes.
- IBM® Rational® ClearCase: Version control, managing changes.
- IBM Rational Requirements Composer: Import requirements to model.

9.2 Extensions

RSA features can be extended using extensions such as the following:

- Extension for integrated frameworks: Leverage Integrated Architecture Frameworks and integration with requirements management to represent business concerns.
- Extension for deployment planning: Topology templates and deployment models.
- Extension for SOA WebSphere: Delivery of Java Enterprise Edition solutions, SOA solutions, and solutions targeted to IBM WebSphere Application Server and IBM WebSphere Portal Server.
- Extension for C++: Include C++ implementations.
- Extension for communication applications: Delivery tools for solutions that include Communications Enabled Applications (CEAs) and integration of CEAs with back-end operational and billing systems (OSS/BSS).
- Simulation Toolkit: Test and validate your architecture to ensure high quality applications, by providing advanced capabilities for requirements identification and validation, design review and validation, and front-end quality assurance