

# Migrating to Rational Systems Developer, Part 1: Rational Rose Models

Skill Level: Intermediate

[Katrina Williams \(kmwillia@us.ibm.com\)](mailto:kmwillia@us.ibm.com)  
IBM Rational Channel Enablement Specialist  
IBM

31 Jan 2006

IBM Rational Systems Developer provides an easy-to-use Eclipse-based environment for modeling and designing your systems. Learn how to successfully migrate your existing IBM Rational Rose models to Rational Systems Developer. This tutorial uses the sample Rose model provided with Rational Systems Developer to show you how to migrate your existing UML models created in Rational Rose into the new Eclipse-based Rational Systems Developer environment.

## Section 1. Before you start

### About this tutorial

This tutorial uses the sample IBM® Rational® Rose model provided with IBM® Rational® Systems Developer along with some lessons learned from actual migration experiences to guide you through the process of migrating your UML models and Java/C++ code from Rational Rose into Rational Systems Developer. Model import utilities are provided with Rational Systems Developer to automate much of the migration process. However, successful migrations require some pre-migration steps to prepare your Rational Rose models. The guidance provided in this tutorial will help to ensure that your migration efforts will be successful.

This tutorial is the first in a two-part series covering the migration of existing UML models and Java/C++ code into Rational Systems Developer. This first tutorial specifically focuses on migrating existing Rational Rose UML models and any

associated code. The second tutorial in the series focuses on the migration of existing IBM® Rational® Rose XDE UML models.

### Tell everyone. Submit this to:



[Digg](#)



[Slashdot](#)

## Prerequisites

Familiarity with Rational Rose and Eclipse and/or Rational Systems Developer is assumed. A working knowledge of UML is also assumed.

## System requirements

In order to complete this tutorial, you should have the following installed:

- Rational Systems Developer v6.0.1.1

**Note:** This tutorial can be applied using Rational Systems Developer running on any of the support platforms.

---

## Section 2. A new paradigm

Rational Systems Developer employs a different paradigm than Rational Rose. Using Rational Rose, you were likely accustomed to "round trip engineering". In round-tripping, the model generates much of the code, particularly those aspects that are structural and repetitive in nature. The developer then goes in and adds algorithmic details directly in the code. These additions are carefully placed between certain specifically formatted comments that the generator can recognize. The generator then re-imports the code, identifies any code changes made that affect the design models, and synchronizes such changes.

Rational Systems Developer operates a bit differently. At first, modeling takes place as it does in round-tripping, where the user specifies models to a degree of detail so that the design level of the code can be fully generated into a compilable state. And as before, developers then go in and flesh out the implementation details. But at this point, when changes to the code go beyond that of implementation details, conventional round-trip tools tend to break down in their effectiveness. The batch nature of having to reverse synchronize design-level changes back into the models can easily become problematic, thus affecting the validity of the very models used to generate the code in the first place.

This is where Rational Systems Developer takes a new approach. After an iteration of modeling work has been taken to the code generation level, the UML visualization features of Rational Systems Developer kick in, instantly rendering the design of existing code into UML form. And when changes happen that affect the design, the UML views are instantly updated - no batch reverse synchronization is needed.

It's important to keep this paradigm shift in mind when preparing to migrate your Rational Rose models. We'll see how this affects the migration as we go through this tutorial.

---

## Section 3. Prepare your workspace

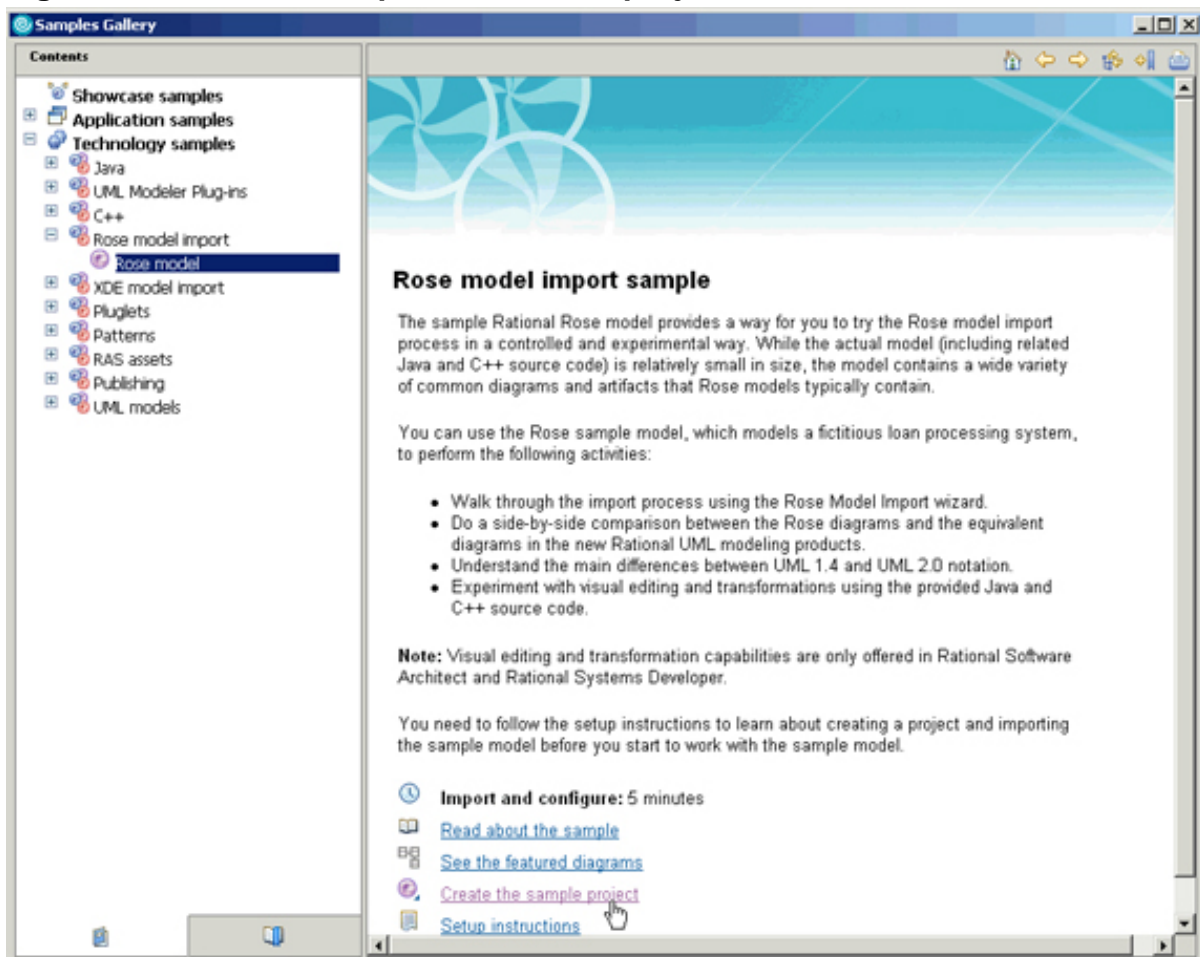
### Setting up the sample project

Before you can import the Rose model that you'll be using in this tutorial, you must create the sample project. The sample project will automatically be created for you when you click the **Create the sample project** link on the **Rose model import** sample page. Perform the following steps to launch Rational Systems Developer, using a new workspace and create the sample project containing the Rose model.

1. Launch **Rational Systems Developer**
2. At the prompt for a workspace, enter a location of your choice to create a new workspace
3. Close the **Welcome window** if it appears
4. Launch the Samples Gallery using **Help -> Samples Gallery**
5. Expand **Technology Samples -> Rose model import**

6. Click on **Rose model** to bring up the Rose model import sample
7. Click on the **Create the sample project** link on the Rose model import sample page
8. Click **Finish** in the **Save Rose model** dialog that appears, accepting the default project name **SampleRoseModel**
9. At the prompt to switch to the Resource perspective, click **Yes**

**Figure 1. Create the sample Rose model project**



Two sample models (RoseImportSample.mdl and RoseImportSample\_custom.mdl) and related Java and C++ source code are added to your workspace and appear in the Navigator view. The models cannot be opened in their current format in Rational Systems Developer. But you can now import them into Rational Systems Developer using the Rose Model Importer.

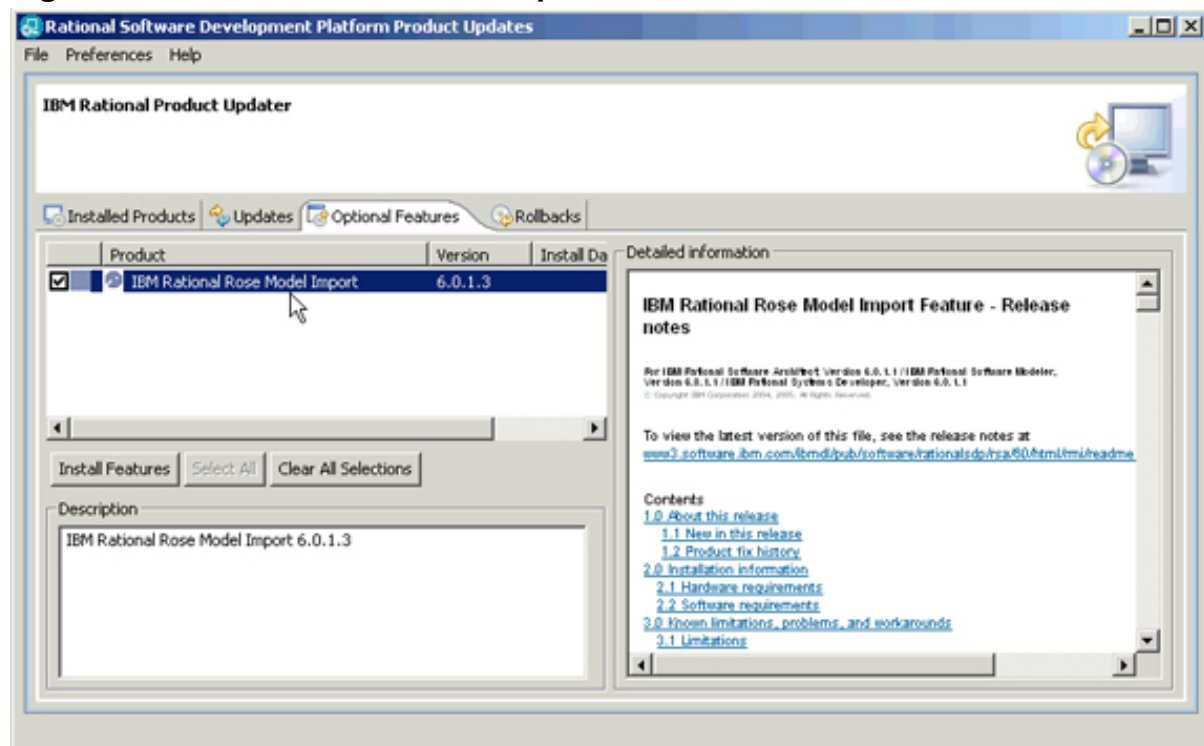
## Installing the Rose Model Importer

The Rose Model Importer for Rational Systems Developer is an Optional Feature that must be installed using the Rational Product Updater. For more information and instructions for installing this optional feature, refer to the [IBM Rational Rose Model Import Feature - Installation and update instructions](#).

If the Rose Model Importer feature is not already installed in your Rational Systems Developer workbench, perform the following steps to install it. **[Note: You must have a live internet connection in order to perform these steps.]**

1. Click **Help -> Software Updates -> IBM RSD Product Updater** to launch the RSD Product Updater tool
2. In the **RSD Product Updater** tool, select the **Optional Features** tab
3. Check the box labeled **IBM Rational Rose Model Import** and Click on the **Install Features** button to install this feature
4. When you complete the installation of this feature, you may be instructed to **restart the workbench**

**Figure 2. Install the Rose Model Importer feature**



---

## Section 4. Importing the Rose model

Once you have set up the sample project containing the Rose model that we'll use for this tutorial and you've installed the Rose Model Importer feature, you are ready to begin importing the sample Rose model. Because the sample Rose model is a very simple model, it will be a very simple import. However, a typical Rose model is not going to be this simplistic. So as we go through the import process using this simple model, I'll be pointing out areas where you will probably need to do some pre-import preparations to your Rose model. Things like controlled units and stereotypes will need to be carefully considered prior to importing.

There is additional information available to you via the **Online Help** section, titled **Installing and migrating**, that you will certainly want to review prior to and during your own model migration efforts. However, this tutorial will give you an idea of the steps involved and some of the things you'll need to plan for during your own migration efforts.

### Specifying the import source

Because the Rose Model Importer needs access to the Rose model being imported as well as to all related controlled units, it's best to perform the import procedure on the same machine that you used to create the Rose model. That way, the Rose Model Importer will have access to the registry and any Rose add-in information, such as stereotype configuration files used by the model.

Because the sample Rose model we are importing is a very simple model, we won't need to worry about specifying a registry file.

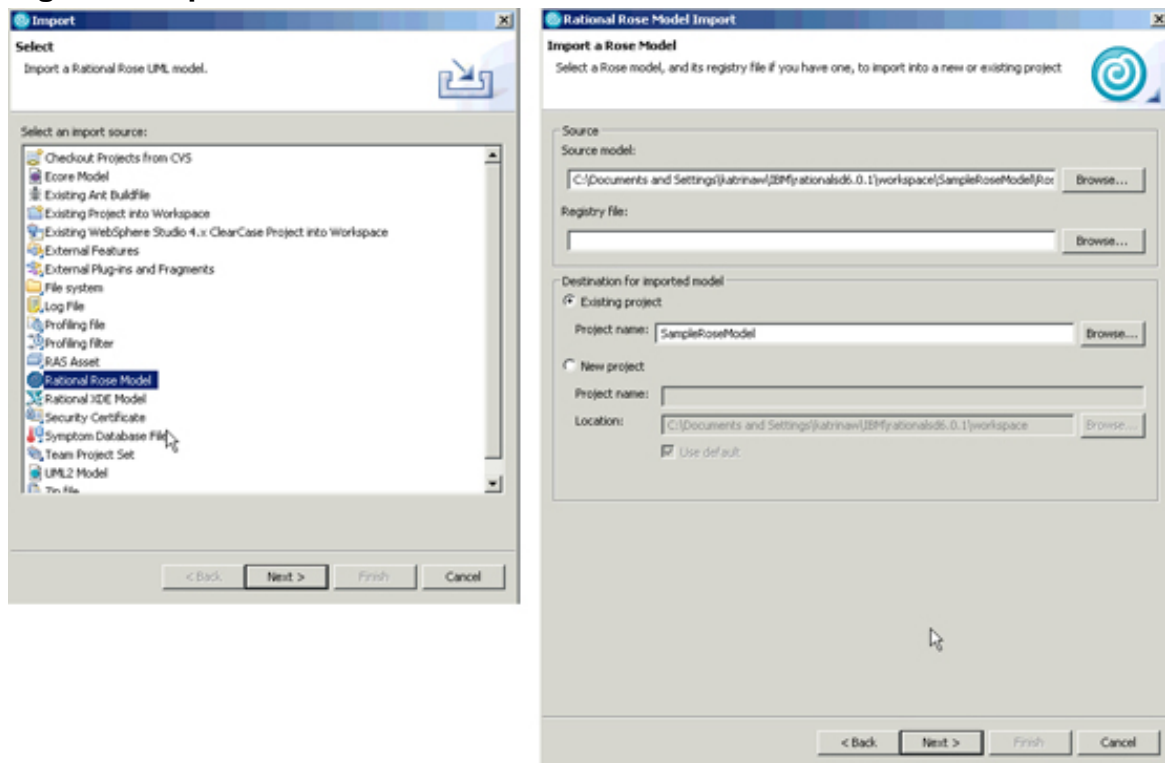
Perform the following steps to begin the import of the `RoseImportSample.mdl` Rose model.

1. In the Navigator view, select the **RoseImportSample.mdl** model
2. Click **File -> Import**. The Import window opens
3. Select **Rational Rose Model**, and then click **Next**
4. On the **Import a Rose Model** page of the **Rational Rose Model Import** wizard, click the **Source model Browse** button
5. Browse to and select the **RoseImportSample.mdl** model, and then click

## Open

6. Verify that the **SampleRoseModel** project is specified as the existing project, and then click **Next**

**Figure 3. Import a Rose Model**



## Resolving subunits and pathmap symbols

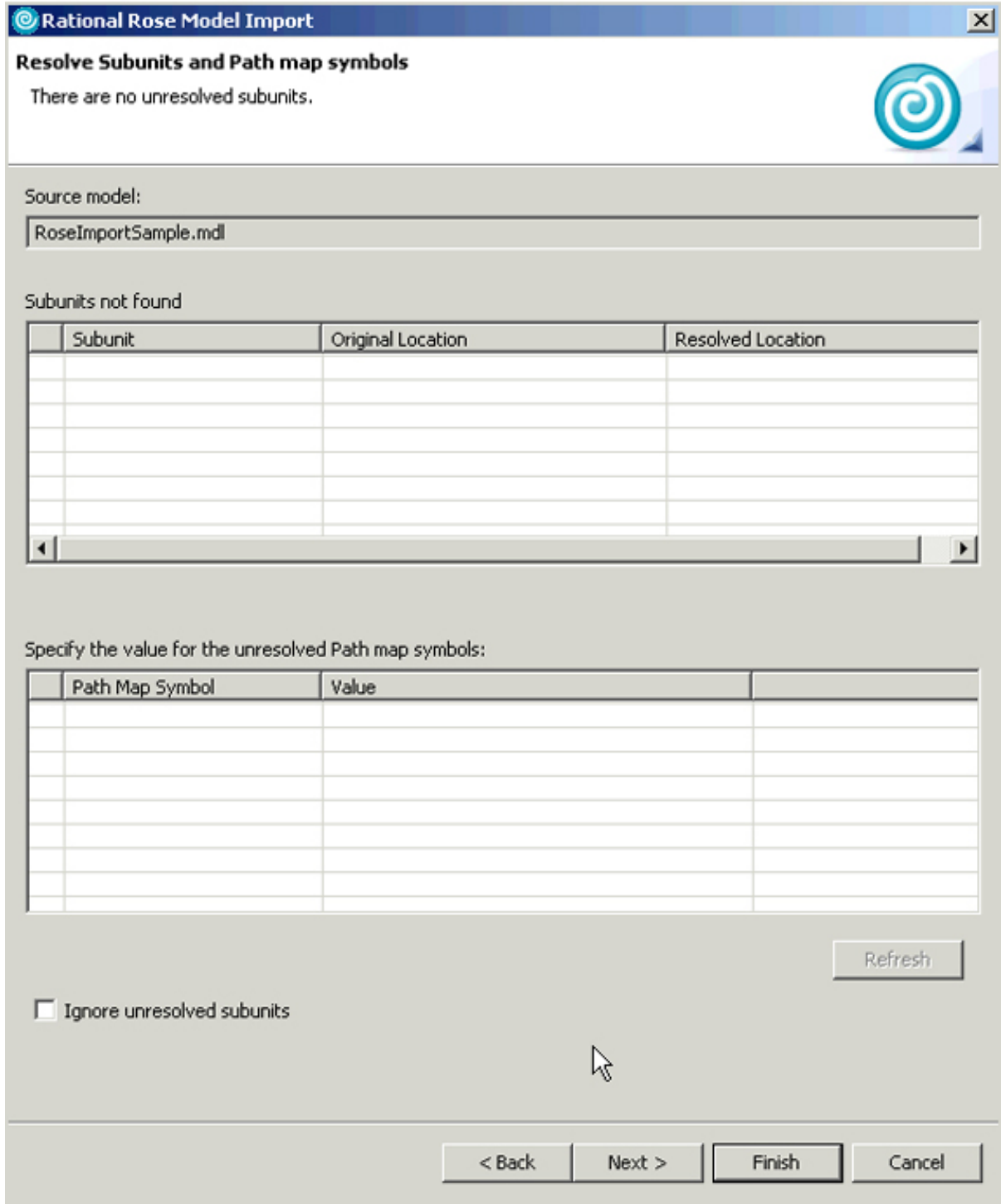
In Rational Rose, a large model can be used by a team to model their system. In such situations, the model is usually divided into individual controlled units (subunits), where each analyst/developer is responsible for one to many subunits. One pre-migration step you should consider in order to avoid potential problems at this point during the import process is to verify that you have access to all of the related subunits.

### Pre-migration tip #1

Open the model in Rose and load all associated subunits to verify that they are all available on the machine from which the import will be performed. It's also a good idea to run a Rose Check Model and fix some of the more blatant errors in the source model. If not, all those problems that you ignored in Rose are going to be migrated as well.

The next dialog that appears allows you to verify and correct any subunit locations and pathmap symbols that may be unresolved. Since our simple sample Rose model does not use any subunits or pathmaps, you can simply click **Next** to continue past this dialog.

#### **Figure 4. Subunits and pathmaps**



## Subunits vs. multiple models

The model management approach in the new IBM UML modeling products, such as Rational Systems Developer, is a major change from Rose. In the new UML

modeling products, you manage and control your models at the model level only; the concept of subunits is not carried over from Rose. While the new UML modeling products do not use subunits, it does not mean that the information stored in subunits is lost upon import.

When you import a Rose model that references subunits, you have two options to consider; you can choose to have packages (CAT) and component packages (SUB) subunits import as separate models, or you can choose to have all subunit content import as packages inside the imported model.

While the model management capability between each tool is different, you can still achieve the same results by dividing your model at the model level instead of at the subunit level.

For more information on the different model approaches between Rose and Rational Systems Developer, see the online Help topic **Installing and Migrating -> Importing Rational Rose models -> Rose model migration -> Key differences for Rose users -> Multiple model approach versus Rose subunits.**

### **Pre-migration tip #2**

When planning the migration of a Rose model that contains controlled units, it is important to think about how you want to partition the resulting Rational Systems Developer model since Rational Systems Developer does not support controlled units. You may want to consider breaking up the imported model into several models. Don't feel constrained by the previous organization of subunits, as the former subunits may be a too fine grain division (or it may be just right). You should be sure to plan for time to document the new model structure and describe it's intended usage after the migration.

The next dialog that appears allows you to choose between importing the Rose model as one monolithic model (incorporating all subunits back into the main model) or importing each subunit as a separate model in Rational Systems Developer. Because our sample model contains no subunits, simply click **Next** to proceed to the next dialog.

### **Figure 5. Import Rose Subunits as Independent Models**

The screenshot shows the 'Rational Rose Model Import' dialog box. The title bar reads 'Rational Rose Model Import'. The main heading is 'Import Rose Subunits as Independent Models'. Below this, there is a sub-heading: 'Select a project to import subunits as independent models or Select a project to map subunits to existing models'. The dialog is divided into two main sections. The first section, 'Create independent models from rose subunits', is selected with a checked checkbox. It contains a 'Destination project for subunit files' group with two radio buttons: 'Existing project' and 'New Project'. The 'New Project' option is selected. Below these are fields for 'Project name' and 'Location' (with a 'Browse...' button), and a checked 'Use Default' checkbox. The second section, 'Map subunits to existing models', is unselected. It contains a 'Source Directory' group with an 'Existing project' field and a 'Browse...' button. At the bottom of the dialog are four buttons: '< Back', 'Next >', 'Finish', and 'Cancel'. A mouse cursor is pointing at the 'Next >' button.

## Rose property sets

**Rose model properties** are modifiable values that you can attach to a Rose model and related model elements. Property sets provide a way for you to define specific

information that is not expressed in standard UML notation, but that is necessary to extend the UML to meet domain-specific requirements of your code, project, or processes.

Unlike Rose stereotypes, Rose model properties do not appear in the diagram editor. Specifically, property sets are implicitly applied (similar to required stereotype extensions in UML) and their names do not appear inside stereotype brackets. Rose model property set files have a .pty file name extension.

A **profile** is a package that contains UML stereotypes that extend the elements (metaclasses) of the UML language for a specific domain or purpose. UML stereotypes have properties that are typed by the standard UML primitive types or by enumerations and classes also defined in the profile. They may also have icons and scalable shape images associated with them. UML stereotypes combine all of the features of Rose stereotypes and property sets into a single metamodel extension mechanism.

Profiles identify a particular subset of the UML metamodel and define stereotypes and constraints that may be applied to it. The UML 2.0 Basic, Intermediate, and Complete, profiles are automatically applied to every model.

UML 2.0 profile files have an .epx file name extension. You can view profiles in the Model Explorer view.

Consider which Rose properties it makes sense to convert to Rational Systems Developer profiles. The criteria may be: Does it include attributes that will be needed after the import? Does it include stereotypes and stereotype icons that will be useful after the import? If not, consider not importing the properties into Rational Systems Developer.

When you do import the properties that you decided to keep, be careful when you fill out the property page in the import wizard. Ignore the properties you do not want to keep, reference the profiles you already created in previous imports, and create the new profiles for the properties that are new for this model into the location where you want them to be. Note that the default on this page is to ignore the properties. See Migrating from Rational Rose -> Migrating models -> Importing a Rational Rose model -> Migrating Rose model property sets to profiles in the Online help for details on importing properties.

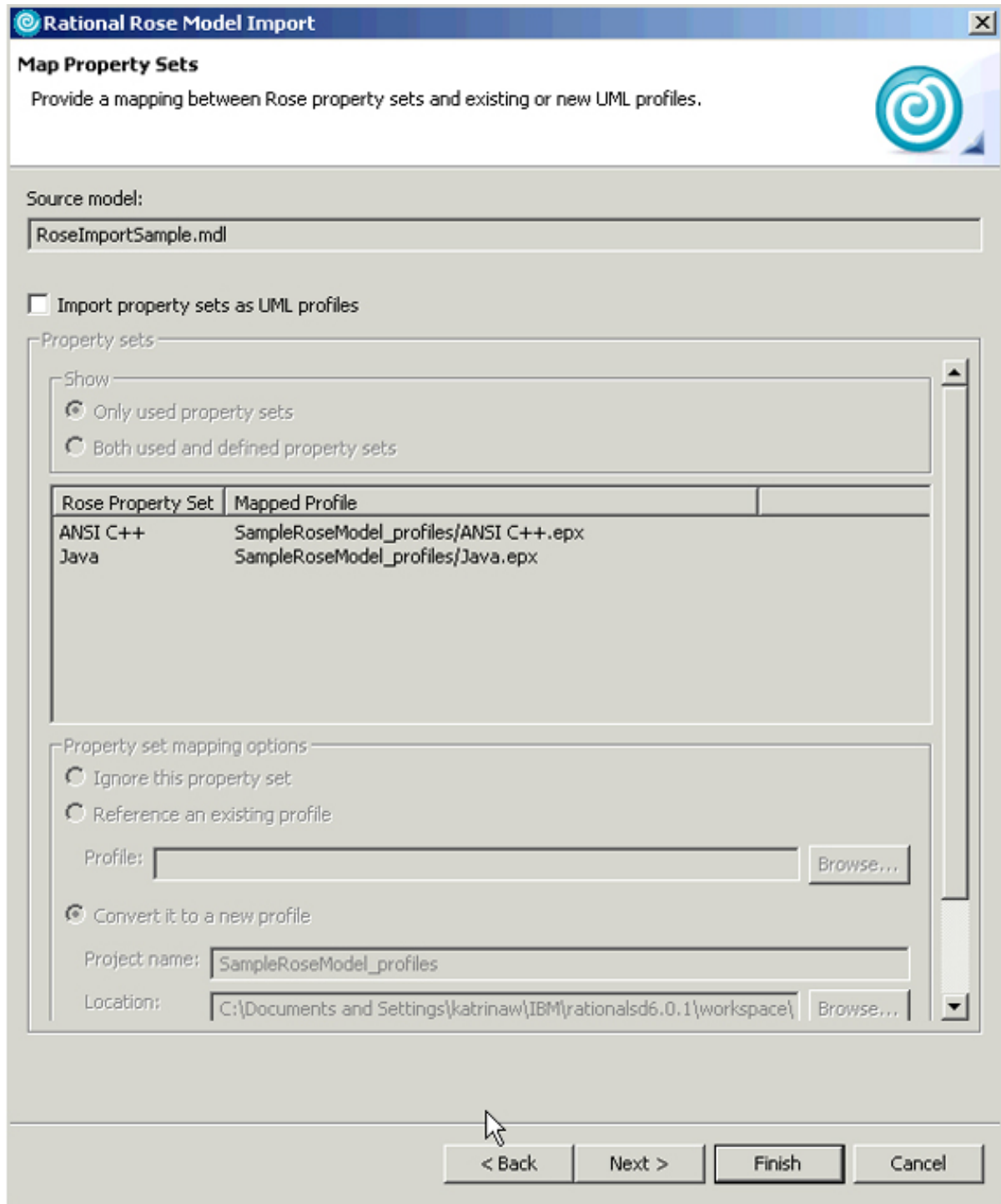
### **Pre-migration tip #3**

If you have properties that are common to several models, or if you want to reduce the memory usage during import, and if you do not override any of the common property values, then you may want to consider importing the model properties separately.

1. In Rose, turn on the addins with the properties you are interested in, create a new empty model in Rose and save it
2. Create a new project in Rational Systems Developer, call it something indicating it contains shared profiles
3. Import the empty Rose model (with the properties) into the Rational Systems Developer shared profile project, make sure you check the profiles to be imported on the profile page in the import wizard, and that you specify that new profiles should be created
4. Now import your real model(s) and on the profile page in the import wizard make sure that you check the box stating that you will refer to your already imported profile.

Note: If a model uses specific model properties, those model properties must be imported at the same time as, or before, the model is imported.

### **Figure 6. Map Property Sets**



## Rose stereotypes

You can map Rational Rose stereotypes (defined in stereotype configuration files) to new or existing profiles using the Map Property Sets page of the Rational Rose

Model Import wizard. You can also choose to skip specific Rose stereotypes during the import process if they are not important or used in your model.

If you do not want to import any of the Rose stereotypes to a profile, clear the **Import stereotypes as UML profiles** check box, and then click **Next** to specify preferences that define how diagram elements appear after import.

#### Pre-migration tip #4

To retain stereotype icons in diagrams, either use the same computer for Rose as well as Rational Systems Developer, or use the following procedure to make the stereotype artifacts available during import.

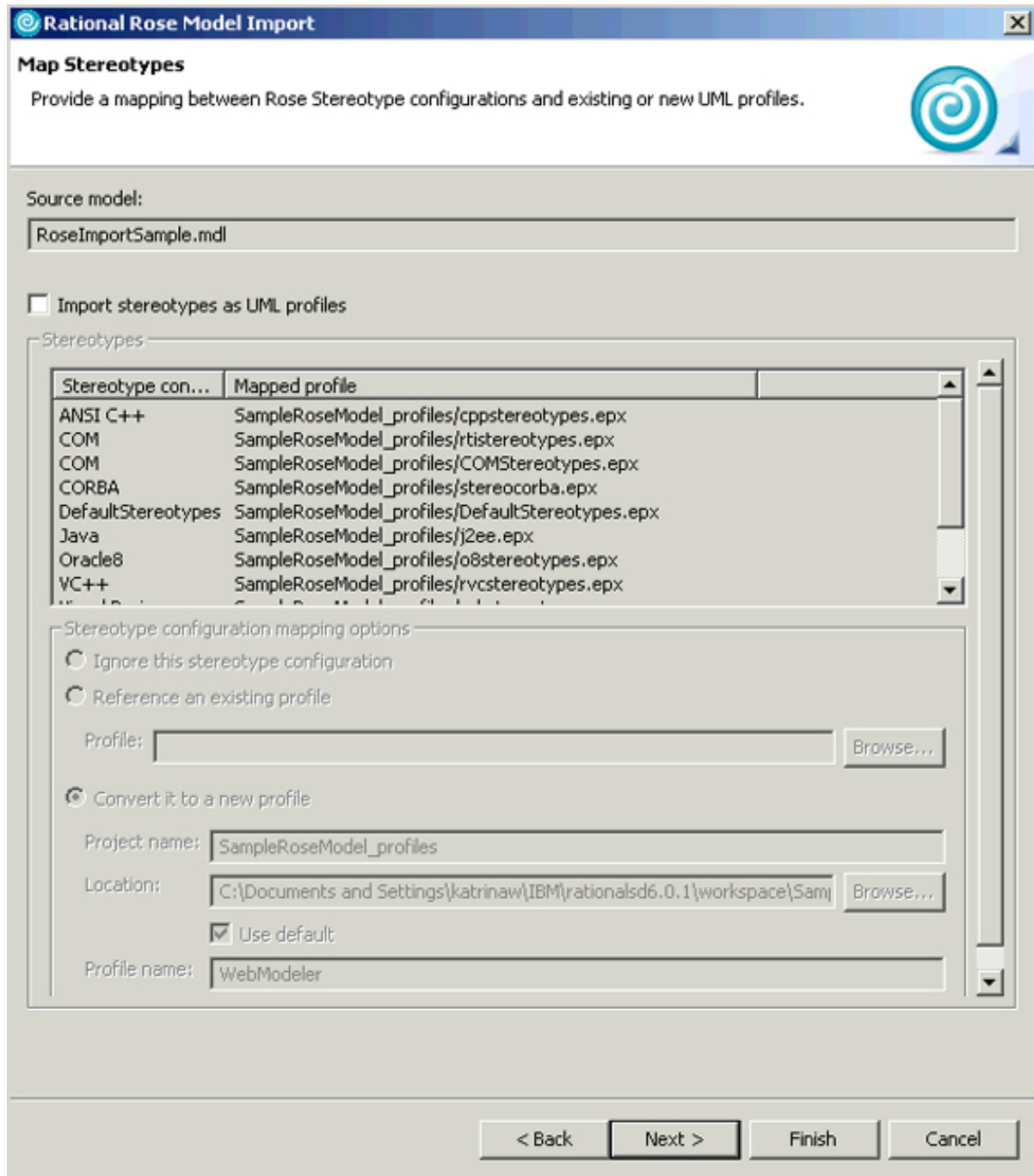
1. Export the Rose data from the Windows or MainWin registry of a machine that has Rose installed and transfer the resulting file to the computer with the new UML modeling product, Rational Systems Developer, installed. Use the following steps to export the Rose registry keys:
  1. Use the regedit or regedt32 tool to export the following registry key:  
HKEY\_CURRENT\_USER\Software\Rational Software\Rose
  2. Name the file rose\_user.reg
  3. Using regedit or regedt32, export the following registry key:  
HKEY\_LOCAL\_MACHINE\SOFTWARE\Rational Software\Rose
  4. Name the file rose\_local.reg
  5. Combine the files, rose\_user.reg and rose\_local.reg, using one of the following methods:
    - On Windows, from the command line, type: `C:\> copy rose_user.reg + rose_local.reg rose.reg`
    - On Windows, in Notepad, open both files. Copy the contents of one to the end of the other and save the result. Be sure to save it with the Unicode encoding, specified in the **Save As** window. Name the file, rose.reg
    - On Linux or Unix platforms, from a command line type: `$ cat rose_user.reg rose_local.reg > rose.reg`
2. From the same Rose computer, copy the stereotype configuration INI files from the base Rose installation and all installed addins, plus all of the associated image files, to the computer that is running Rational Systems Developer. Be careful to maintain the identical Rose directory structure because the registry and stereotype INI file information depends on that

same structure.

3. In the Rational Rose Model Import wizard, specify the registry export file created in the first step in the **Registry file** field on the first page of the wizard
4. Proceed through the wizard to the **Map stereotypes** page and ensure that all of the stereotype configuration files obtained in step 2 appear and are mapped to the existing or new UML profiles
5. Finish the wizard to complete the import.

For our sample import, we'll **clear** the check box labeled **Import stereotypes as UML profiles** to avoid importing any stereotypes and then simply click **Next** to continue.

### Figure 7. Map Rose Stereotypes



## Rose model preferences

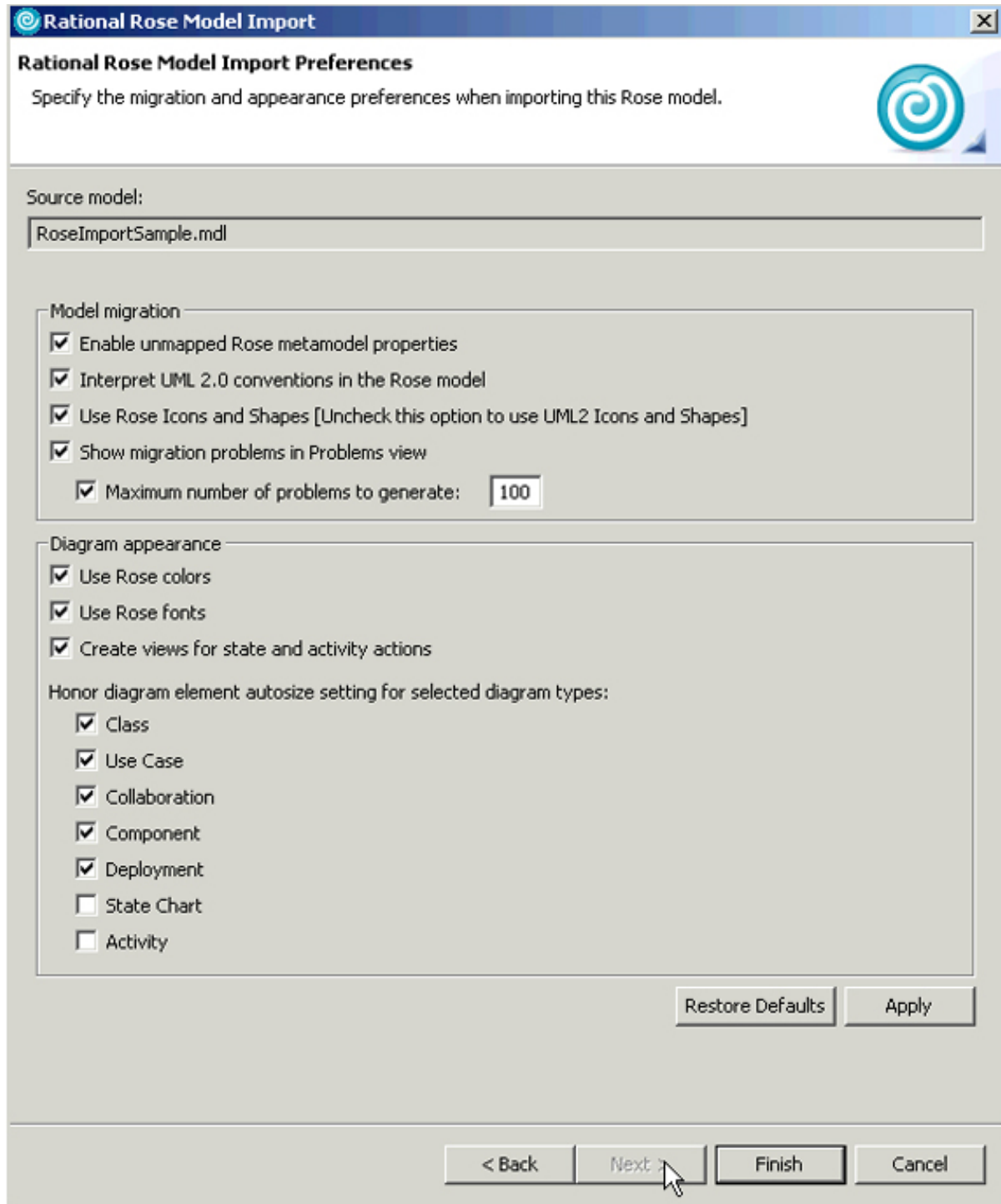
You can specify the appearance of the imported model using the Rational Rose Model Import Preferences page of the Rational Rose Model Import wizard.

Refer to the Rational Rose Model Import Preferences page context-sensitive help

(F1) for detailed information on all of the preferences you can specify.

Click **Finish** to complete the import process.

**Figure 8. Rose Model Preferences**



---

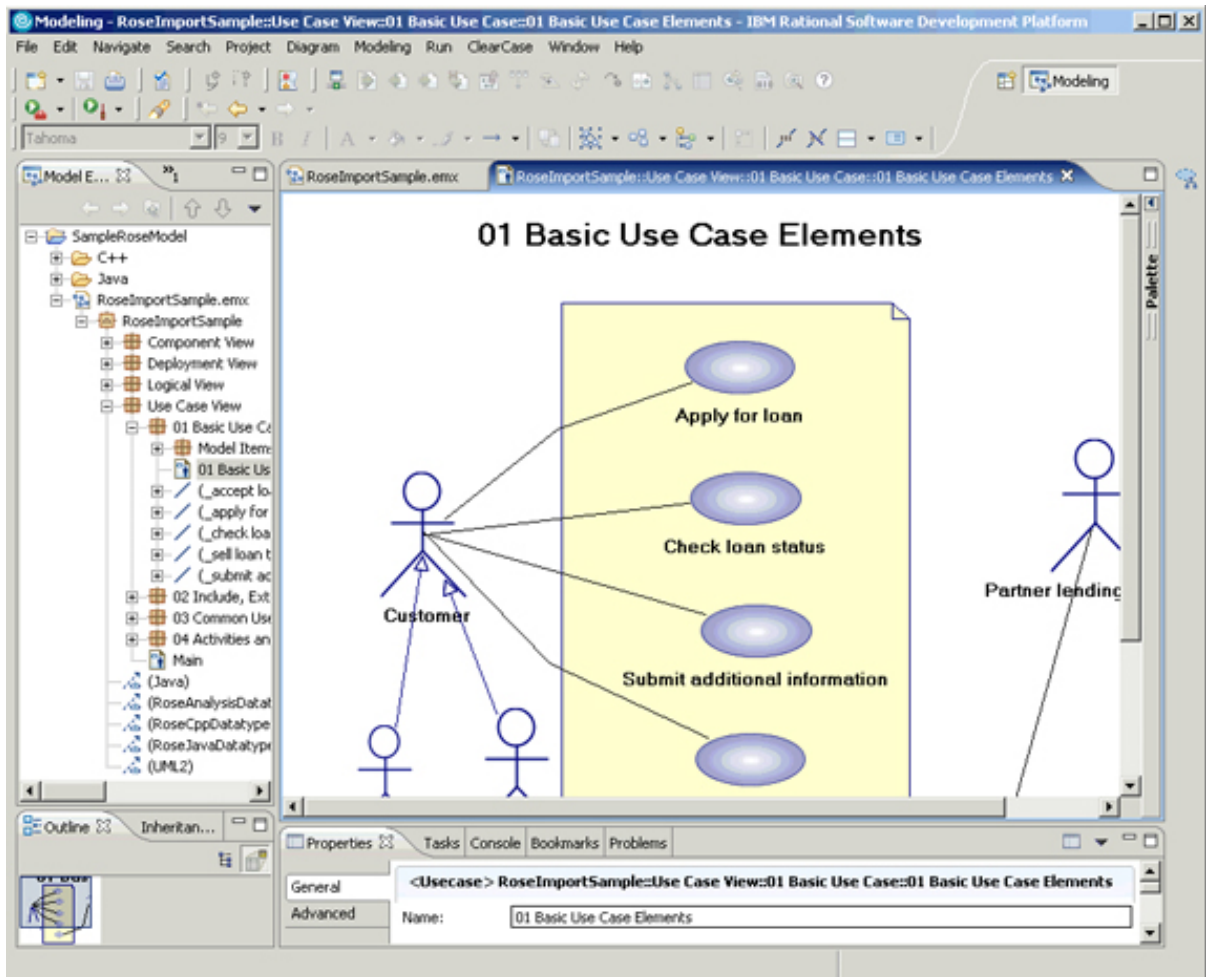
## Section 5. Examining the results

Now the fun begins! Let's see what was created by the Rose Model Importer for us. The first thing you'll notice is that the same package structure was created as what we had in Rose, including the Use Case View, Logical View, Component View and Deployment View. Of course, you can rename these packages or create new packages and move your model elements if you wish. However, the best approach (particularly when working with very large Rose models) would be to delete any elements that are not necessary for the migration to Rational Systems Developer. I'll discuss that in more detail as we examine the resulting Rational Systems Developer model.

To begin, let's open the **01 Basic Use Case** package inside the **Use Case View** package. Double-click on the **01 Basic Use Case Elements** diagram to open it and see that it looks similar to what you are familiar with in Rose. Use Case diagrams are pretty straight-forward. They migrate into Rational Systems Developer with little change or differences.

There is little difference between Use Case diagrams using UML 1.4 (which is what Rose was based upon) and UML 2.0 (which is used in Rational Systems Developer). So for the most part, the diagrams should migrate and look the same. There are, however, significant changes in the area of sequence diagrams, state diagrams, activity diagrams and collaboration diagrams that you'll want to be aware of. If you have made use of any of these diagrams in Rose, you'll want to study the Online Help topic **Installing and migrating -> Importing Rational Rose models -> UML 1.4 and 2.0 notation mapping** to familiarize yourself with the differences and prepare to closely examine and perhaps "touch-up" the diagrams that are created in Rational Systems Developer during the migration.

### Figure 9. Examining the new Use Cases



Once you've spent some time perusing through the Use Case View package to examine its contents and familiarize yourself a bit with the differences between UML 1.4 and UML 2.0, let's move onto another area of significant difference between Rational Rose and Rational Systems Developer. Recall the **New Paradigm** highlighted at the beginning of this tutorial: Rational Rose Round-tripping vs. Rational Systems Developer Transformations. To fully understand how this will impact your Rose migration efforts, expand the **Logical View -> 06 Implementation Model** package that was created in Rational Systems Developer from the Rose model import.

You'll notice that the two packages **C++** and **Java** were created just as they existed in the Rose model. And in those packages are the *implementation-level* class specifications that were used in Rational Rose to generate the corresponding code. Or, they may even have been created in Rose by reverse-engineering the code. Either way, this level of detail is neither necessary nor recommended in the "Model Driven Development" environment of Rational Systems Developer. Rather than import the Implementation Model from Rose, it is recommended that you actually delete that package from the Rose model prior to import. Often times, the

implementation models developed in Rational Rose quickly become out-of-synch with the actual code. And rather than having to go through the effort of reverse-engineering the code to create an accurate implementation-level model of the code in Rational Rose and then import that into Rational Systems Developer, we can simply bring the actual code into Rational Systems Developer and use the UML visualization capabilities in it to render our code into accurate UML representation. That is the topic of the next section of this tutorial.

### Pre-migration tip #5

When migrating a Rose model to Rational Systems Developer, delete the parts of the Rose model that are at a "code-level" of abstraction. These parts include classes, operations and attributes that directly correspond to code. Migrate only the parts of the Rose model that are at a higher level of abstraction (e.g., architectural and high-level design content).

---

## Section 6. Importing and visualizing the code

Using the **Rose Model Import** feature provided with Rational Systems Developer, we have at this point successfully imported our Rational Rose UML model into Rational Systems Developer. At this point, you would probably be spending some time comparing the model in Rational Systems Developer with the original model in Rational Rose and cleaning up any formatting issues in the diagrams. You'll also want to run a **Validation** on the Rational Systems Developer model just to identify any errors that may have cropped up. The typical types of errors that may be identified are related to adherence to UML 2.0 requirements, logic, structure, and use. All results of the validation, such as errors, warnings, and informational messages, appear in the **Problems** view. You'll also want to spend some time reviewing the help topics on Rose model element mapping to understand the differences in UML notation. Most of the UML differences are somewhat minor. However, as was stated earlier, the major differences are found in state machines, activities and interactions.

What you do next greatly depends upon what you were using Rational Rose for. If you were using it specifically as a UML modeling and design tool, then your migration work is probably complete at this point. However, if you were using it to generate Java or C++ code from your UML models, or using it to reverse engineer your Java or C++ code to generate UML diagrams from your code, then you'll need to bring your existing code into Rational Systems Developer and begin making use of the UML visualization feature. This section of the tutorial focuses on the process of importing and visualizing your Java/C++ code. The process of importing and visualizing code is pretty much the same for both Java and C++ (except for the type

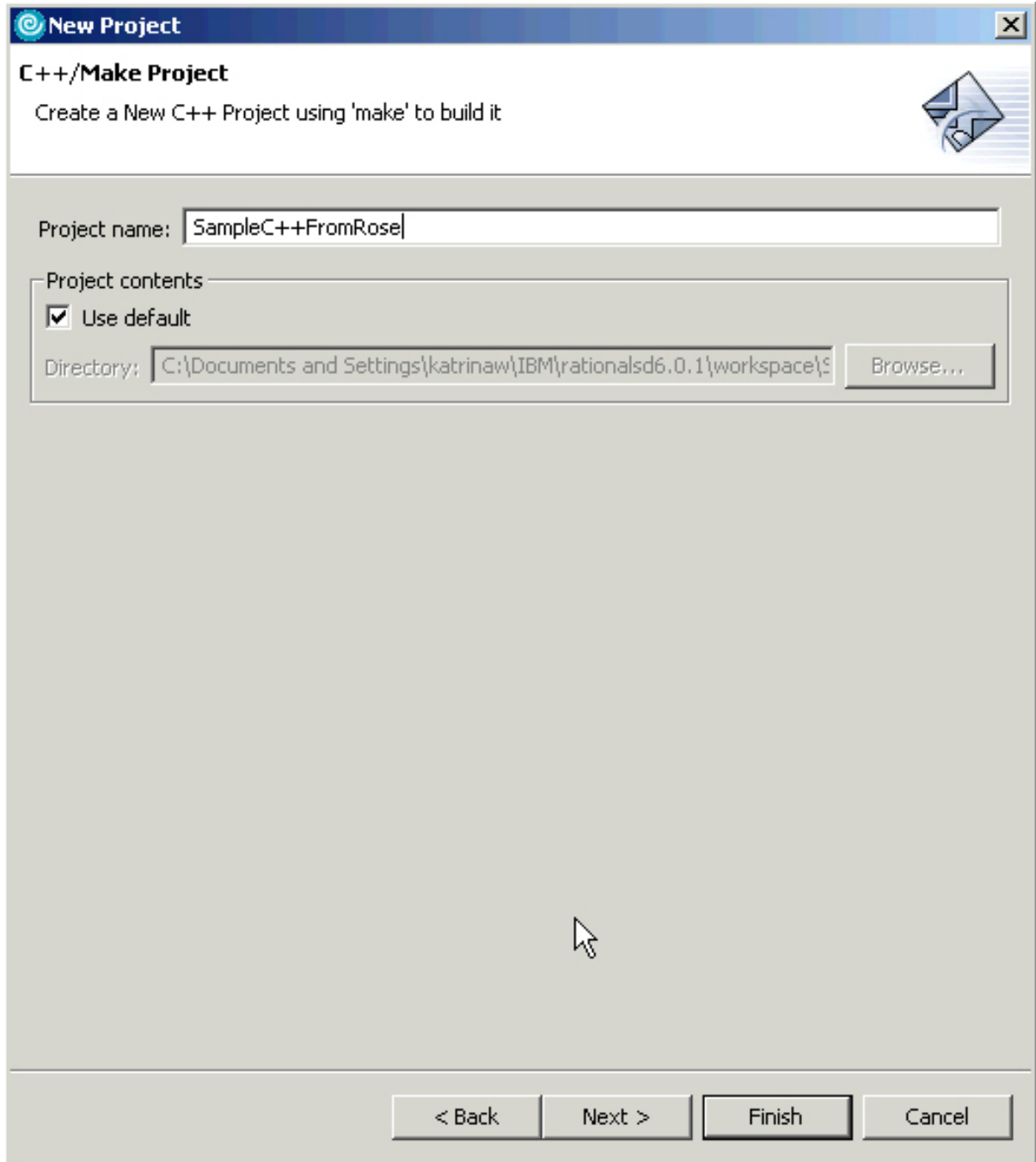
of project you create to hold the code), so for the purposes of this tutorial, we'll import the C++ code supplied with the sample Rose model.

Before you can import code into Rational Systems Developer, you will want to create a project specifically for holding the code. This tutorial will not go into the specifics of the types of projects available to you with Rational Systems Developer. If you are not already familiar with the concept of projects and of the different types of projects available with Rational Systems Developer, refer to the Online Help as there is quite a wealth of useful information found there.

Because we'll be importing C++ code, you'll need to create a C++ project in your Rational Systems Developer workspace by performing the following steps.

1. On the main menu bar, select **File -> New Project**
2. In the **New Project** dialog, expand the **C++** folder
3. Select the **Standard Make C++ Project** option and click **Next**
4. Provide a project name, and click **Finish** to create the project

#### **Figure 10. Creating a C++ Project**

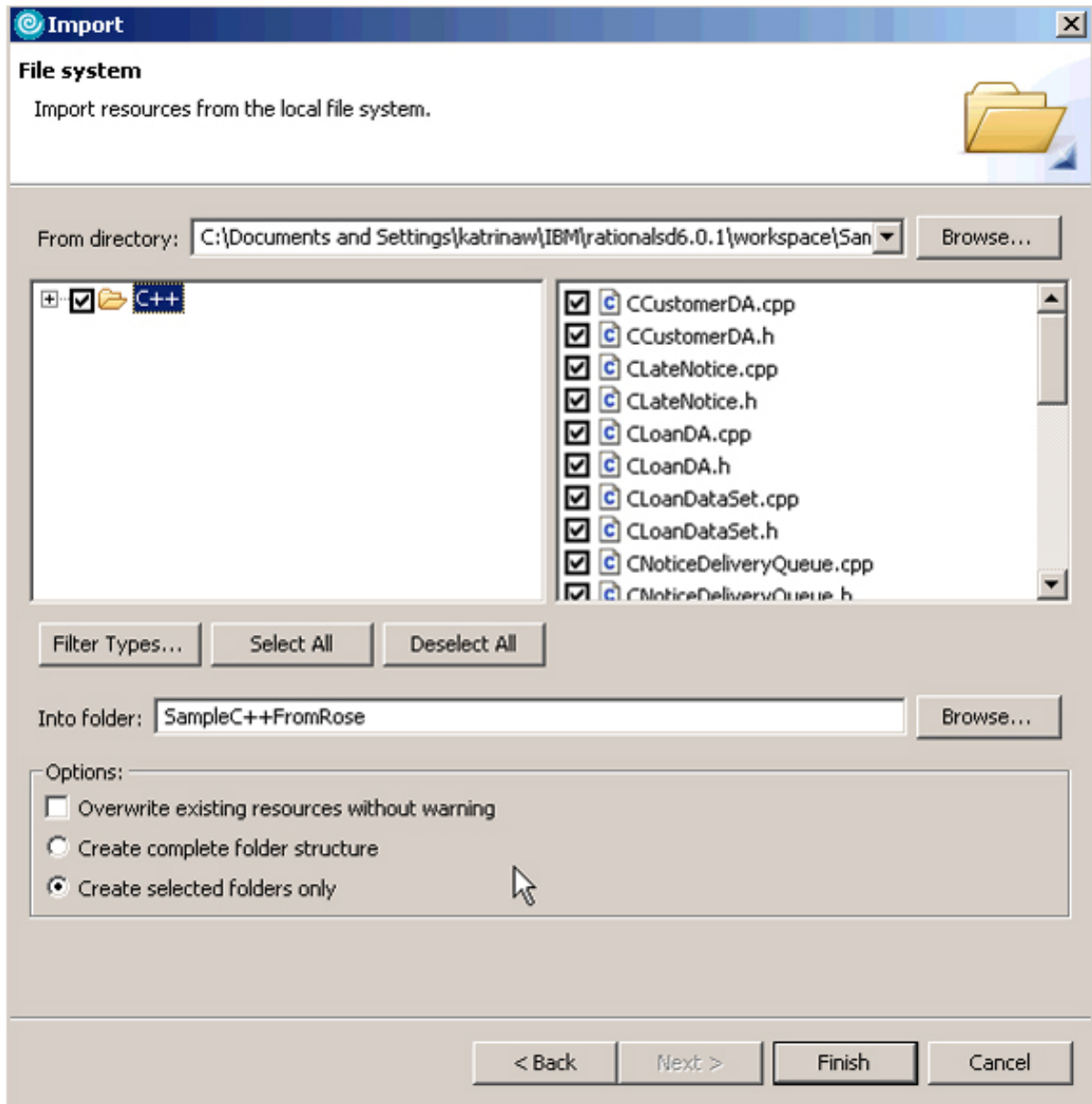


Great! Now you've got a C++ project into which you can import the C++ code. Just like with the Rose Model Import, there is a wizard for importing the C++ code from the file system. Follow the steps to import the C++ code into your new C++ project.

1. From the main menu, select **File -> Import**
2. In the Import dialog, select **File system** and click **Next**

3. In the source dialog that appears next, browse to the location of the sample C++ code that is provided with the sample Rose model. It should be located in **SampleRoseModel/C++** subdirectory of your workspace.
4. Check the box next to the **C++** folder in the left pane to cause all of the code to be selected, verify the location in the **Into folder** field to be the C++ project you created, and click **Finish**

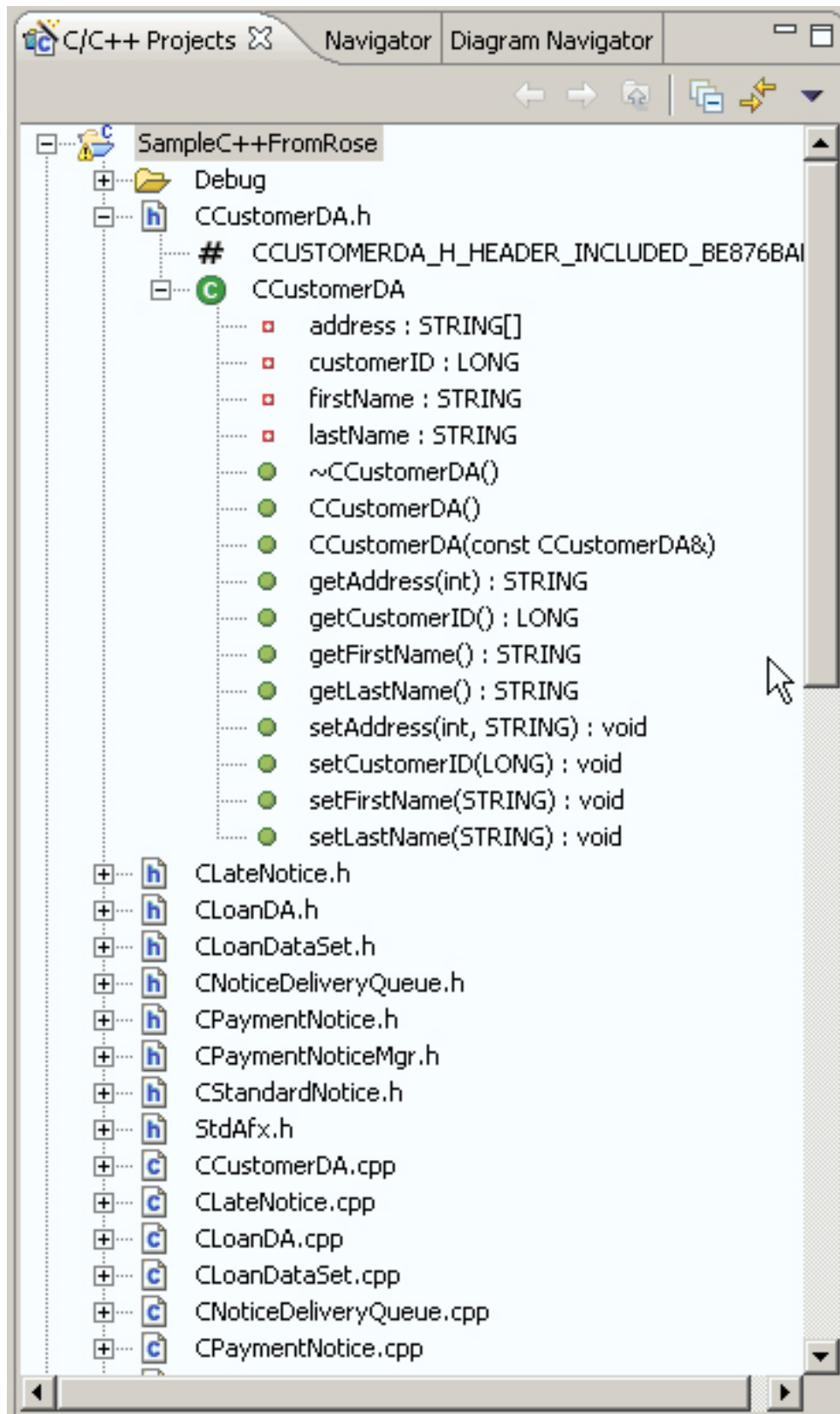
**Figure 11. Importing the C++ Code**



Now let's take a look at what was created during the import process. Right away we can see the benefits of using Rational Systems Developer to work with our C++

code. The Model Explorer view allows us to see our methods and fields with icons. This graphical view of the C++ code makes it much easier to quickly see what our code looks like without having to actually look at the code itself. However, with Rational Systems Developer we are, in fact, looking at the code itself. This is not a separate implementation model we're looking at, but rather the code depicted in a more visual way.

## **Figure 12. Examining the Results**

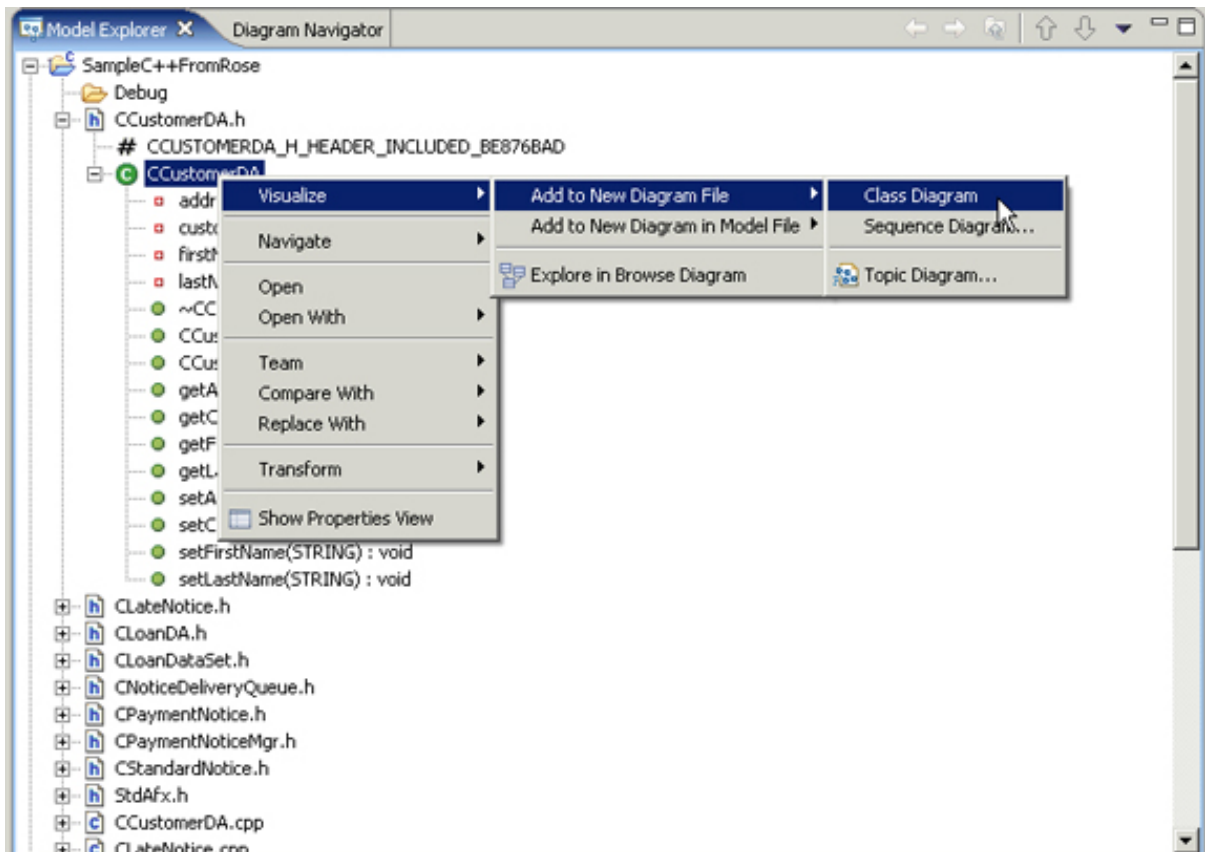


As you can see, the explorer/navigator view provides a nice way to see what methods and fields are associated with our code. But it doesn't show the relationships or interactions between our code. Recall that in Rose we were accustomed to creating detailed class diagrams and sequence diagrams in our UML model and eventually generating code from the model. Or, if the code already existed, we could reverse-engineer the code to get a UML representation of it in the form of a class diagram. But with Rose, that diagram and model were separate from the code. As the code changed, a manual effort was required to synchronize the model with the updated code. And, as the model was updated the synchronization would have to occur again to update the code. With this manual effort, it was very often the case that once our projects got to the implementation phase we rarely took the time to go back and update our UML models.

Rational Systems Developer eliminates this problem by allowing us to visualize the code in a UML diagram without creating a separate "instance" of the code. When we visualize the code in a Class diagram, we are actually looking at the code itself rather than a copy of it. Let's create a class diagram using the visualization capability of Rational Systems Developer to see what it looks like.

1. Right-click on **CCustomerDA**
2. Select **Visualize -> Add to New Diagram File -> Class Diagram**

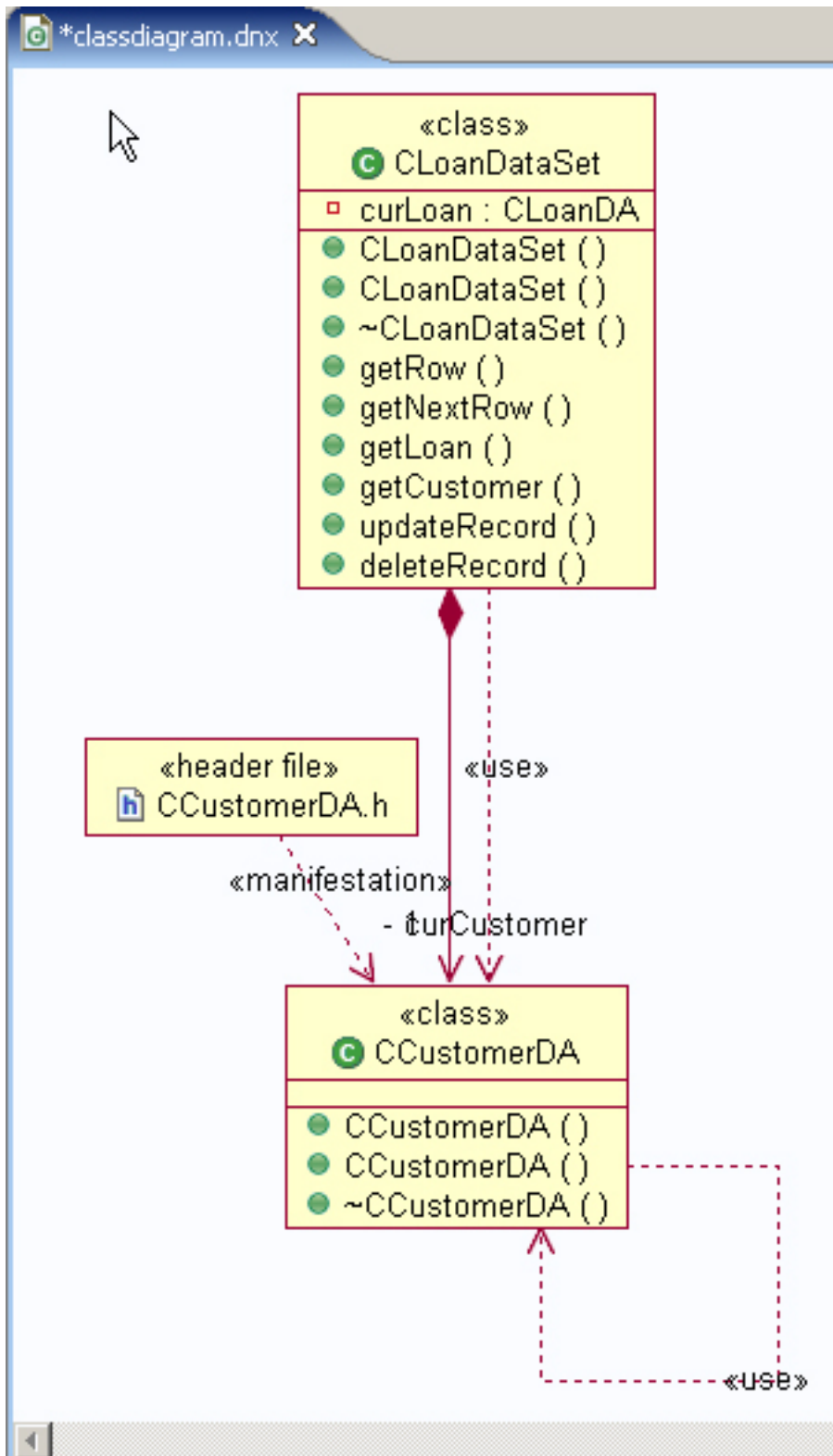
### Figure 13. Visualize the code



A new class diagram is created displaying the selected class. This diagram can be modified and any modifications made (e.g., adding a method) will automatically cause the underlying code to be updated. You can easily navigate to the code by double-clicking on the class on the diagram. You can also display related elements and other information using the context menu accessed by right-clicking on the class on the diagram. Let's use this context menu to display related elements.

1. Right-click on the **CCustomerDA** class on the diagram
2. From the context menu, select **Filters -> Show Related Elements**
3. Select **Show All Relationships** and click **OK**

**Figure 14. Visualize the code**



## Section 7. Summary

This tutorial stepped you through the process of migrating a UML model from Rational Rose to Rational Systems Developer. While the Rose Import Wizard is fairly straight-forward to use and provides you with quite a few options to consider while importing your model, what you do prior to importing the model can be even more important. Following these tips should make your migration efforts smooth and successful.

- Open the model in Rose and load all associated subunits to verify that they are all available on the machine from which the import will be performed. It's also a good idea to run a **Rose Check Model** and fix some of the more blatant errors in the source model. If not, all those problems that you ignored in Rose are going to be migrated as well.
- When planning the migration of a Rose model that contains controlled units, it is important to think about how you want to partition the resulting Rational Systems Developer model since Rational Systems Developer does not support controlled units. You may want to consider breaking up the imported model into several models. Don't feel constrained by the previous organization of subunits, as the former subunits may be a too fine grain division (or it may be just right). You should be sure to plan for time to document the new model structure and describe its intended usage after the migration.
- If you have properties that are common to several models, or if you want to reduce the memory usage during import, and if you do not override any of the common property values, then you may want to consider importing the model properties separately. Then you'll just need to reference the shared profile project on the profile page of the Rose Model Import wizard.
- To retain stereotype icons in diagrams, either use the same computer for Rose as well as Rational Software Modeler, Software Architect, or Systems Developer, or follow the recommended steps to stereotype artifacts available during import.
- When migrating a Rose model to Rational Systems Developer, delete the parts of the Rose model that are at a "code-level" of abstraction. These parts include classes, operations, and attributes that directly correspond to code. Migrate only the parts of the Rose model that are at a higher level of abstraction (e.g., architectural and high-level design content).

# Resources

## Learn

- [IBM Rational Rose Model Import Feature - Installation and update instructions](#)
- [The new IBM Rational design and construction products for Rose and XDE users](#)
- [Model Structure guidelines for Rational Software Modeler and Rational Software Architect](#)
- [IBM Rational Systems Developer product page](#): Find technical documentation, how-to articles, education, downloads, and product information about Rational Systems Developer.
- Stay current with [developerWorks technical events and Webcasts](#).

## Get products and technologies

- Build your next development project with [IBM trial software](#), available for download directly from developerWorks.

## Discuss

- [Participate in the discussion forum for this content](#).
- [Rational Software Architect, Software Modeler, Systems Developer, Application Developer and Web Developer forum](#): Ask questions about Rational Systems Developer.
- Participate in [developerWorks blogs](#) and get involved in the developerWorks community.

## About the author

### Katrina Williams

Katrina is a Channel Enablement Specialist with the Rational brand in IBM. Katrina spent 10+ years developing software for US Navy and Army simulators working first as a software developer and later designing and supporting Software Development Environments as a System Administrator/CASE Tool Administrator for IBM and Lockheed Martin. Katrina has experience using Rational brand products as a customer (while working for IBM and Lockheed Martin) and as a Customer Support Engineer working for Rational Software. Katrina now supports the IBM Rational sales organization providing training materials, demonstrations, and presentation materials for the design and construction products.