This third tutorial in this series covers how you can use an external systems model as part of the bottom-up track of the "meet-in-the-middle" approach. It continues using the online DVD rental case study introduced in the previous two parts, and you will use IBM® Rational® Software Architect to produce an external system model based on that case study.

Section 1. Before you start

Learn what to expect from this tutorial and how to get the most out of it.

About this series

This tutorial series gives you a detailed look at modeling service-oriented architecture (SOA) by using IBM® Rational® Software Architect. Although primarily for software architects and about the activities that they perform, it is also helpful to people in other roles in the software development process, including those who provide input into software architecture, such as business analysts, and those who use the software architecture as input to perform their own activities, such as software designers and developers (architecture realization, design, and implementation). This series also covers several core SOA concepts that are useful to a wide audience.
These tutorials focus on three topics:

- **Architecture**: Describe what the architecture comprises and where it fits into the overall software development process.
- **Services**: Create the architecture for an SOA system (services are central to this architecture).
- **Models**: Demonstrate how the Rational Software Architect supports a model-driven development (MDD) approach to the specification of service-oriented architectures.

After describing software architecture and defining the place of services within it, this series then introduces Rational Software Architect and its SOA- and architecture-related features. By using a fictitious online DVD rental case study throughout, these three tutorials:

- Describe the work products used as input to the service architecture activities, including the component business model, business process model, system use case model, and external systems part of the design model.
- Describe, step-by-step, how the service model representing the architecture is specified in Rational Software Architect, including service consumers, service specifications, service partitions, atomic and composite service providers, services, service collaborations, service interactions, and service channels.
- Explain how the service model is then used in the subsequent phases of the software development process, such as design and implementation.

**About this tutorial**

In **Part 1**, we introduced the video rental case study that is used as the example throughout this tutorial series. We placed service architecture within the framework of the Rational Unified Process and introduced the IBM SOA Solution Stack for reference. We noted the various work products that are used as input to a service architecture, and then used the case study to provide examples for two of them: the business architecture model (described in Part 1 in the form of a component business model) and the business process model.

In **Part 2**, we took a detailed look at what a domain model is and how it can be represented in Rational Software Architect. You started to get hands-on experience with the tool and created the domain model used in this series.

In this part, we'll cover how you can use an external systems model as part of the bottom-up track of the "meet-in-the-middle" approach.

**Objectives**
After completing this tutorial, you should be able to:

- Describe how the external system model is used to model external software systems
- Produce an external system model for the case study

Prerequisites

To get the most value from this tutorial, it is recommended (but not necessary) that you be familiar with:

- Service-oriented architecture (SOA)
- IBM Rational Software Architect V7.0 (fix 002 recommended) or later
- Unified Modeling Language (UML)
- IBM® Rational® Method Composer®, previously known as IBM® Rational Unified Process® (RUP®)

We highly recommend reading the first two parts in this tutorial series before reading this part.

System requirements

Rational Software Architect Version 7 (FixPak 003 recommended) or later.

---

Section 2. Positioning external systems and bottom-up analysis

Part 1 of this tutorial series mentioned the external systems model as an input to the service architecture activity. The brief description given was this: The non-SOA systems that you can leverage. As we noted previously, the external systems model can be used as part of the bottom-up track of our meet-in-the-middle approach.

Next, we explain why having a view on the non-SOA systems that you leverage as part of the solution important in an SOA-style integration project, along with how this is achieved by using the external systems model.

Positioning the external systems model

The external systems model does not formally exist in RUP. However it does map
onto an existing RUP work product: The design model. Figure 1 attempts to illustrate how this mapping works.

**Figure 1. Mapping the external system model to the design model**

Using Figure 1 as reference, we define the following about the external system model:

- It captures, at a design level, a representation of non-service-oriented software that forms a part of the solution.
- The software that it covers can include both software that is new to the business (for example, new packages that are being purchased and require integration), as well as existing software used by the business (commonly known as legacy software), although the focus is still on non-service-oriented software.
- Its purpose is to capture the form and constraints of software that is external to but that is integrated into service-oriented software systems.
- As it maps to the design model, we it is "owned" by the RUP Analysis and Design discipline.
- The owning role is the designer who would create or source (get) most of the model during the Inception phase in a RUP-based project. We say "create or source" because the designer may be either leveraging existing model elements and diagrams or creating new ones.
- We include the external system model as one of the work products created as part of the bottom-up track of the meet-in-the-middle approach.

The service model is a design-level model, as well. When the distinction is made between a service model and a design model, the service model will contain the specification of the design at a higher level of abstraction and use only SOA elements (service provider, service, service consumer, service specification, for example). The design model then acts as a detailed design model and contains both SOA and non-SOA artifacts.
In practical terms, the external system model can be seen either as a set of packages within the design model containing the information, as scoped in Figure 1, or as totally separate from the design model but at the same level of abstraction.

**Note:**
We use the word *system* as it is defined in the Rational Unified Process (see Resources for more information): "A collection of connected units that are organized to accomplish a specific purpose." Wherever we use the word *system*, you can potentially replace it with sub-system, solution, application, composite application, or business application.

**Bottom-up analysis in an SOA or integration project**

In SOA, the bottom-up approach is about analyzing existing IT assets (such as legacy applications and systems) and finding functionality that could be exposed as services to be reused for many purposes. For example, the bottom-up approach analyzes existing Information Management System (IMS) transactions or COBOL programs.

Reuse is an important part of SOA and critical to its success. As you probably know, your legacy applications (that is, those that have already been deployed) are your company's most valuable assets, thus it is important to take advantage of them whenever and however you can.

**Note:**
IBM offers methods, techniques, and engagement models specifically for "legacy to SOA transformation." Also, using existing asset-analysis tools, such as IBM WebSphere Studio Asset Analyzer, is critical, because, often, nobody knows exactly what is deployed and running! However, this is not the subject of this article. Here, we just want to show how you would capture the results of a bottom-up approach by using Rational Software Architect.

The bottom-up approach is used on integration projects where some software needs to be integrated into the overall solution and this software has not been designed and built as service-oriented software. By that, we mean software where the parts of the architecture were not specifically designed as interacting service consumers and service providers. This involves more than just simply adding services onto your software parts. Specifically, a service-oriented solution needs its logic factored in such a way that its individual parts can be integrated into a number of service-oriented systems.

Bottom-up analysis is all about investigating the external form of the software that is being integrated into your solution, so that you have a clear understanding of two key things:

- The functionality provided by that software
- Any constraints that this places on the solution
This analysis work is done during the RUP Inception phase for integration projects, because this information may have an important bearing on the cost and schedule of the project.

A handful of examples:

- You may discover that there is software that needs to be integrated into the solution that has considerable complexity in its exposed API (application programming interface).
- There may be functionality that needs to be integrated that was not apparent from the top-down work (in other words, from looking at the business processes).
- In engaging with system specialists for these external systems, you may discover that there are changes being made to their exposed APIs that will affect the timelines of your own project.
- There might be duplication of data between systems that will need to be addressed by the architecture of the solution.
- Where external systems are being provided by third parties and these are new to the organization, the bottom-up analysis might challenge high-level statements made by the third party about the accessibility, completeness, and versioning stability of the API exposed by their system.

**Important:**
It is crucial that the "unknown" risks associated with integrating with an external system are mitigated during Inception, before budgets and time constraints are agreed upon for the project. Otherwise, there might be some nasty surprises in store!

Section 3. Modeling external systems and their interfaces

Creating an external systems model in Rational Software Architect

The starting point here is the SOA tutorial project that is the result of Part 2, which contains our domain model. Download the file (from the Downloads section here), and then follow these instructions to import the project into your workspace.

**Note:**
If you still have the workspace available from Part 2, skip these steps and proceed to
Step 8.

1. Start **Rational Software Architect**. Use the default workspace, or create a new one.

2. After Rational Software Architect has launched, close the **Welcome Screen** if you are in a new workspace.

3. Select **File > Import**.

4. In the **Import wizard**, type **project** in the **Select an import source filter** field, and then select **Project Interchange** and click **Next** (Figure 2).

**Figure 2. Import Project Interchange**

5. Click **Browse** and point to the location where you saved the DVD-Rental-DomainModel-RSA-ProjectInterchange.zip file.
6. Select **SOA Tutorial** and click **Finish** (Figure 3).

**Figure 3. Import the SOA tutorial project**

![Import Project Interchange Contents](image)

7. Select **Window > Open Perspective > Modeling** to switch to the **Modeling** perspective.

8. Expand **SOA Tutorial > Models**, and double-click **Domain Model** to open it. Click **OK** if prompted by a Warning that referenced models are not available.

9. You should see something like Figure 4 in your **Project Explorer** view.

**Figure 4. Initial Project Explorer view**
We will use a section of our design model as our external system model. So to start with, you need to create a new UML model for the design model.

10. Select the **SOA Tutorial** project. Right-click it, and then click **New > UML Model**.

11. From the **New UML Model** dialog, leave the **Standard template** selected and click **Next**.

12. Enter **Design Model** as the file name, and deselect **Create a default diagram** in the model.

13. Click **Finish**.

You will now have an empty design model in your SOA tutorial project.

14. Select the **Design Model**, right-click it, and then click **Add UML > Package**.

15. Name it **External Systems**.

16. Delete the **Main diagram** in this package by right-clicking on it, and then clicking **Delete from Model**. This results in a new model, as shown in Figure 5.

**Figure 5. New external systems package (in the design model)**

![New external systems package](image-url)
Identifying the external systems

Now we need to understand what external systems will be a part of our solution. In modeling the Return Video business process, we had noted that the Retrieve member’s standing automated task needs to be done by integrating with an existing Customer Relationship Management (CRM) system. Assume that your discussion with the business process owner has resulted in you being put in touch with a senior developer who is responsible for the CRM system. He has given you the following information:

- The system is an off-the-shelf package that was purchased from a small CRM vendor several years ago.
- The software runs on a .NET platform, but it has a Web services API exposed.
- There are notes in a PDF document called “Integrating with the CRM System's API” provided with the software. It describes the form of the Web services API, along with instructions about how to invoke it.

This is good, because there should be enough information in the PDF document to populate our external systems model. However this remains to be seen.

1. Create a new package under External Systems for our Customer Relationship Management system. Call it CustomerRelationshipMgt.
2. Right-click on the package in the Project Explorer view, and select Add UML > Subsystem. Name it CustomerRelationshipMgt also. (Make sure that you keep the <<subsystem>> stereotype when you rename it.)
3. Rename the default diagram that was created in the package to CustomerRelationshipMgt ExternalSystemSpec.
4. Make sure that the diagram is open, and drag the new subsystem onto it.

The result of all this is as shown in Figure 6.

**Figure 6. The CustomerRelationshipMgt external system**

Now we’re ready to model our external system.
Identifying the provided interfaces

While reading the "Integrating with the CRM System's API" document, we find out that the system has several Web service-based APIs, each of which exposes a different set of CRM-related functionality. However, we find a specific API called the Customer Functions API that gives us access to the CRM information that we require. Instead of modeling all of the exposed APIs, we will focus on just modeling the API that we require. Based on the information in the document, we create an interface for this API in our model:

1. Create a package under the **CustomerRelationshipMgt** package. Name it Provided Interfaces, and delete the default diagram.

2. Create a package under the **Provided Interfaces** package. Name it **CustomerFunctionsAPI**. Rename the default diagram to **CustomerFunctionsAPI InterfaceSpec**.

3. Right-click on the **CustomerFunctionsAPI** package, and then select **Add UML > Interface**. Name it **CustomerFunctionsAPI**.

4. The **CustomerFunctionsAPI InterfaceSpec** diagram should be open. Drag the new interface onto it.

The diagram should appear as in Figure 7.

**Figure 7. The CustomerFunctionsAPI InterfaceSpec diagram**

We'll look at detailing this interface later (note that, for this system, we have only a provided interface --- there are no required interfaces). For now, we'll link it to our external system spec:

5. Open the **CustomerRelationshipMgt ExternalSystemSpec** diagram.

6. Select the **CustomerFunctionsAPI** interface in the **Project Explorer** view, and drag it onto the diagram that we've just opened.

7. From the **Palette**, find **Realization** under the **Class** section. Click on it, and then click on the **CustomerRelationshipMgt** subsystem on the diagram, and drag a realization to the **CustomerFunctionsAPI** that was
just placed on the diagram.

The diagram should now look like Figure 8.

**Figure 8. The CustomerRelationshipMgt ExternalSystemSpec diagram**

![Diagram](image)

We'll change the way this diagram appears now and add a link to the interface spec diagram:

8. Right-click on **CustomerRelationshipMgt**, Filters > Show External View.

The external view shows the provided interface using what is commonly called the *lollipop* (or *lollypop*) notation, because of the symbol.

9. To remove the **CustomerFunctionsAPI** from the diagram, right-click on it from the diagram, and select **Delete from diagram**.

10. Select the **CustomerFunctionsAPI InterfaceSpec** diagram from the **Project Explorer** view. Drag it onto the **CustomerRelationshipMgt ExternalSystemSpec** diagram that we had open, so that you create a diagram shortcut.

**Tip:**
Adding diagram shortcuts makes the model easier to navigate. This is especially useful when the models are published in HTML format.

11. Click **Note Attachment** in the **Palette** (see Figure 9) and draw a note attachment between the new diagram shortcut that we've created and the lollipop symbol representing the **CustomerFunctionsAPI** on the **CustomerRelationshipMgt** subsystem.

**Figure 9. Creating a note attachment**
Section 4. Modeling external system information and interfaces

As we've learned earlier in Part 2, our domain model provides a structured view of the information that exists in the business domain. This is extremely useful when communicating internally within IT, as well as when IT needs to communicate about the business. However it's important to note that, although the domain model will influence the form of the software solution, it is not modeling software directly.

It is, however, useful to have a domain model-type view of the information that is managed by a specific system. Instead of structuring the terms (domain types) used by the business, it will structure the terms used by the external system interfaces (provided and required). It will specifically focus on those types that are stored and managed by the external system.

This domain model-type view is called an information model. We create an
information model for each external system that persists some information (which is most systems). (We can also use information models within our service model—more on that in a future part of this series.)

Modeling the information view

An information model looks very much like a domain model. The main difference is that this model contains information types (info types) rather than domain types. In many respects, an info type looks exactly like a domain type. The difference is that an info type describes a structured type of information that exists in a software domain (external system or architectural part); whereas, a domain type describes a structured type of information that exists in a business domain.

Looking at the "Integrating with the CRM System's API" document again, we discover that the information types stored by the Customer Relationship Management system that is relevant to the Customer Functions API are customer and customer category. We also note a description of the attributes of both of these types of information, along with a note that a customer is related to a single customer category.

Let's add this information to our model.

1. Create a new package under the CustomerRelationshipMgt package. Name it Info Types.
2. Delete the default diagram and create a new class diagram. Name it CustomerRelationshipMgt InfoTypes.
3. Create the following model elements:
   - An <<infoType>> Customer Class with the following attributes:
     - customerId: String
     - name: String
     - address: String
     - telephoneNumber: String [0..1]
   - An <<infoType>> CustomerCategory with a name String attribute
   - A * to [0..1] association between Customer and CustomerCategory

The result is shown in Figure 11.

If you need help doing this, check Part 2. Modeling the business domain, about creating the domain model. Note that <<infoType>> is a keyword.

Figure 11. CustomerRelationshipMgt InfoTypes diagram
Although we haven't modeled any in this example, as with a domain model, an information model can have enumerations and constraints.

The last thing we'll do with our info model is add a diagram shortcut to our `CustomerRelationshipMgt ExternalSystemSpec` diagram (see Figure 12).

**Figure 12. Adding a diagram shortcut to the information model**

Detailing the interfaces

What we modeled in the previous subsection is a representation of the information that is persisted by the external system. We also need to flesh out the interface specification that we created earlier. We do this by detailing the operations that exist on the interface. The operation's signature is expressed in terms of parameters and, optionally, a return type. These are based on another kind of type: the parameter
Let's look at an example.

Again looking at the "Integrating with the CRM System's API" document, we find a description of the operations that the Web service provides. We'll add this information to our model.

1. Open the CustomerFunctionsAPI InterfaceSpec diagram.
2. Right-click on the CustomerFunctionsAPI Add UML > Operation. Name it retrieveCustomer.

Let's first look at the operation's parameters.

3. Click the retrieveCustomer operation.
4. In the Parameters section of the Properties view, right-click in the empty list, Insert New Parameter (Figure 13).

Figure 13. Insert New Parameter

5. Change Name to customerId, and set Type to String.

We'll now change the diagram so that this information is shown.

6. Right-click on CustomerFunctionsAPI in the diagram, and select Filters > Show Signature.

The CustomerFunctionsAPI InterfaceSpec should appear as in Figure 14.

Figure 14. The retrieveCustomer operation with a parameter added
We'll now define the return type of the operation. This will need to be based on a parameter type. Note that, in our example here, we use a parameter type for the return type. However, parameters can be described in terms of parameter types where required.

7. Create a new package under the `CustomerRelationshipMgt` package, and name it `Parameter Types`.

8. Rename the default diagram to `CustomerRelationshipMgt ParameterTypes`.

9. Create a new class called `Customer` and add the `parameterType` key word to it.

10. Add attributes as shown below in Figure 15.

**Figure 15. The new parameter types package**

11. Open the `CustomerFunctionsAPI InterfaceSpec` diagram.

12. Drag the `Customer` parameter type onto the diagram.

13. Select the `retrieveCustomer` operation.

14. In the **General** section of the **Properties** view, click **Set return type**.

15. Set the return type to `Customer`, thereby ensuring that you select the Customer parameter type that is owned by the `CustomerRelationshipMgt` external system.

The diagram should now appear as in Figure 16.
We've now reached the point where we have a concrete view of the external system that we need to integrate (albeit a very simple example for this tutorial). We have a firm view of what the interface that we will be integrating with looks like, and we also understand what information we are using from the system and how it is structured. Although this is quite a simple example, the same principles are essential when tackling larger external systems and managing their complexity.

Section 5. What's next

In this part of this tutorial series, we looked at the bottom-up track of the meet-in-the-middle approach, and, more specifically, the external non-SOA systems that are its focus. We discussed the importance of this activity in an SOA project, and then we looked, in detail, at how to create an external systems model by using Rational Software Architect. More specifically, we looked at modeling provided interfaces and information models. In the parts of this series that follow, we will get into the core of SOA modeling: Creating the service model.
## Downloads

<table>
<thead>
<tr>
<th>Description</th>
<th>Name</th>
<th>Size</th>
<th>Download method</th>
</tr>
</thead>
<tbody>
<tr>
<td>project</td>
<td>DVD-Rental-DomainModel-RSA-ProjectInterchange.zip</td>
<td>9KB</td>
<td>HTTP</td>
</tr>
<tr>
<td>project</td>
<td>DVD_Rental-ExternalSystemModeling-RSA-ProjectInterchange.zip</td>
<td>13KB</td>
<td>HTTP</td>
</tr>
</tbody>
</table>

Information about download methods
Resources

Learn

- A four part tutorial series by the same author: Design SOA services with Rational Software Architect
- In the Pattern Solutions area on developerWorks, get the resources you need to advance your skills in patterns-based development.
- Read the developerWorks introductory article, "The Rational UML profile for business modeling" by Simon Johnston (April 2004). The Rational UML profile for Business Modeling is a component of the Rational Unified Process (RUP). It presents a UML language for capturing business models and is supported by the Business Modeling Discipline in the RUP.
- Read the developerWorks intermediate-level article, "Business services modeling, Integrating WebSphere Business Modeler and Rational Software Modeler" by Jim Amsden (December 2005). Business Services Modeling forms the foundation for the integration between IBM WebSphere Business Modeler, Rational Software Architect (and UML), and the Rational Unified Process (RUP) business modeling guidelines to better support model-driven development (MDD).
- Visit the Rational software area on developerWorks for technical resources and best practices for Rational Software Delivery Platform products. In particular, visit the Rational Software Architect area.
- Subscribe to the developerWorks Rational zone newsletter. Keep up with developerWorks Rational content. Every other week, you’ll receive updates on the latest technical resources and best practices for the Rational Software Delivery Platform.
- Browse the technology bookstore for books on these and other technical topics.

Get products and technologies

- Download a trial version of Rational Software Architect.
- Download trial versions of IBM Rational software.
- Download IBM product evaluation versions and get your hands on application development tools and middleware products from DB2®, Lotus®, Rational®, Tivoli®, and WebSphere®.

Discuss

- Rational Software Architect, Data Architect, Software Modeler, Application Developer and Web Developer forum: Ask questions about Rational Software
Architect.

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

About the authors

Gregory Hodgkinson
Gregory Hodgkinson is founder, director, and the SOA lead at 7irene, an IBM Tier 1 Business Partner in the United Kingdom (www.7irene.com). He has 10 years of experience in software architecture, initially specializing in the field of component-based development (CBD), then moving seamlessly into service-oriented architecture (SOA). His extended area of expertise is the software development process, and he assists 7irene and IBM customers in adopting RUP framework-based agile development processes and SOA methods. He is still very much a practitioner, and has been responsible for service architectures for a number of FTSE 100 companies. He presents on agile SOA process and methods at both IBM (Rational and WebSphere) and other events. He has also co-authored a Redbook on SOA solutions.

Bertrand Portier
Bertrand Portier is an IT Architect with SOA Advanced Technologies, IBM Software Group. He works in the field on strategic SOA transformation projects and, based on these experiences, works with IBM Software Group development teams. His background is in J2EE and Web services and he is now heavily involved with asset-based and model-driven development.