Modeling Service-Oriented Architectures

An Illustrated Example using Sparx Systems Enterprise Architect

by Doug Rosenberg

Chapter 1

A Roadmap for Service-Oriented Architecture Development using Enterprise Architect Business and Software Engineering Edition

Trying to make sense of the acronym soup that engulfs important topics like software architecture, business modeling, and service oriented architectures is a major challenge. We're going to take a shot at it in this book by following a single example all the way from architecture to code. The example is a Service-Oriented Architecture (SOA) car rental system that's implemented with a combination of web-services and custom software.

Along the way we'll illustrate many of the key features of *Enterprise Architect Business and Software Engineering Edition*. As with a number of other topics we've addressed in our books, we're going to use a Process Roadmap to tie everything together in what we hope will be a clear and understandable manner.

Why a Service-Oriented Architecture Example?

We didn't have to choose a Service-Oriented Architecture for our example project; SOA is one of many possible architectures that can be developed using Enterprise Architect's Business and Software Engineering Edition. But web services and SOA have become increasingly important in today's IT universe. Recent estimates suggest that nearly half the companies in the world are adopting, piloting, or considering SOA. With those sorts of numbers, it stands to reason that many readers will be interested in a roadmap and a cohesive example that brings some clarity to the problem. Hopefully this includes you.

As it turns out, a SOA example also serves to illustrate many of the features of the Sparx Systems solution, which supports building "executable business processes" that use WSDL (Web Service Definition Language) to implement their solution. And, since projects don't live by web services alone, Enterprise Architect has numerous other useful features for handling those parts of the application that require custom (non-web service) development. In particular we'll spend a significant amount of time in this book exploring a new and unique capability for ***behavioral code generation*** for the enforcement of Business Rules, and tight integration with IDEs, including Eclipse and Visual Studio. We've leveraged the power of these capabilities into our SOA Roadmap.

A Quick Introduction to SOA

Service-Oriented Architecture (SOA) is an approach to building complex software systems from a set of reusable services that obey service-orientation principles. Many people believe that SOAs can help businesses to be more “agile” -- in other words, enable faster and more cost-effective responses to changing conditions.

SOA enables construction of applications from fairly large chunks of reusable functionality that can be built quickly, primarily from existing services. Thus, SOA promotes reuse at a “macro” level, and, in theory, as an organization publishes more and more business functionality as services over time, the cost of building applications that use those services decreases.

A service that obeys the principles of service-orientation is an *autonomous, loosely coupled*, and *stateless* unit of functionality that is made available by a *formally defined interface*. The functionality provided by a service is *discoverable* by applications that use the service. In other words, services expose their functionality via interfaces that other applications and services can read to understand how to use them. Development of services can thus be decoupled from development of the applications that use them. The ***Universal Description Discovery and Integration*** (UDDI) specification defines a mechanism to publish and discover information about services.

Services are not allowed to call other services, but may communicate with them via messages. That is, services are loosely coupled, and each service implements a single action, such as placing an online rental car reservation. Services are also loosely coupled to underlying operating systems and insulate application code from specific technologies that are used to implement the services.

Services are designed without knowing who will be using them. In a service-oriented architecture, applications are built from services which communicate via messages using a process known as ***orchestration***.

A popular approach to implementing a service-oriented architecture is via web services, which make services accessible over the Internet independent of platforms and programming languages. The BPEL language is often used to orchestrate SOA applications. Building applications from services requires metadata that describes the characteristics of the services, and the data that is used. Typically, XML is used to describe data, WSDL describes the services themselves, and SOAP (Simple Object Access Protocol) describes communication between services.

Two important roles in developing service-oriented applications are that of the service provider and service requester. Service providers develop their services and publish them to a broker, while service requesters use the brokers to locate services, bind to them, and then use them.

The SOA Roadmap (aka "ICONIX Process for SOA Development")

As with all of our ICONIX Process Roadmaps[1], our SOA roadmap is defined as a set of activity diagrams. In this case, the roadmap provides a "cookbook" set of steps that can be followed for building systems that are based around an SOA.

Figure 1-1 shows our top-level roadmap. The chapter outline of this book follows this roadmap directly. Each of these top-level activities will expand out to a child diagram showing further detail. Those detailed activities will be discussed in the respective chapters – this chapter gives you the "big picture" overview.

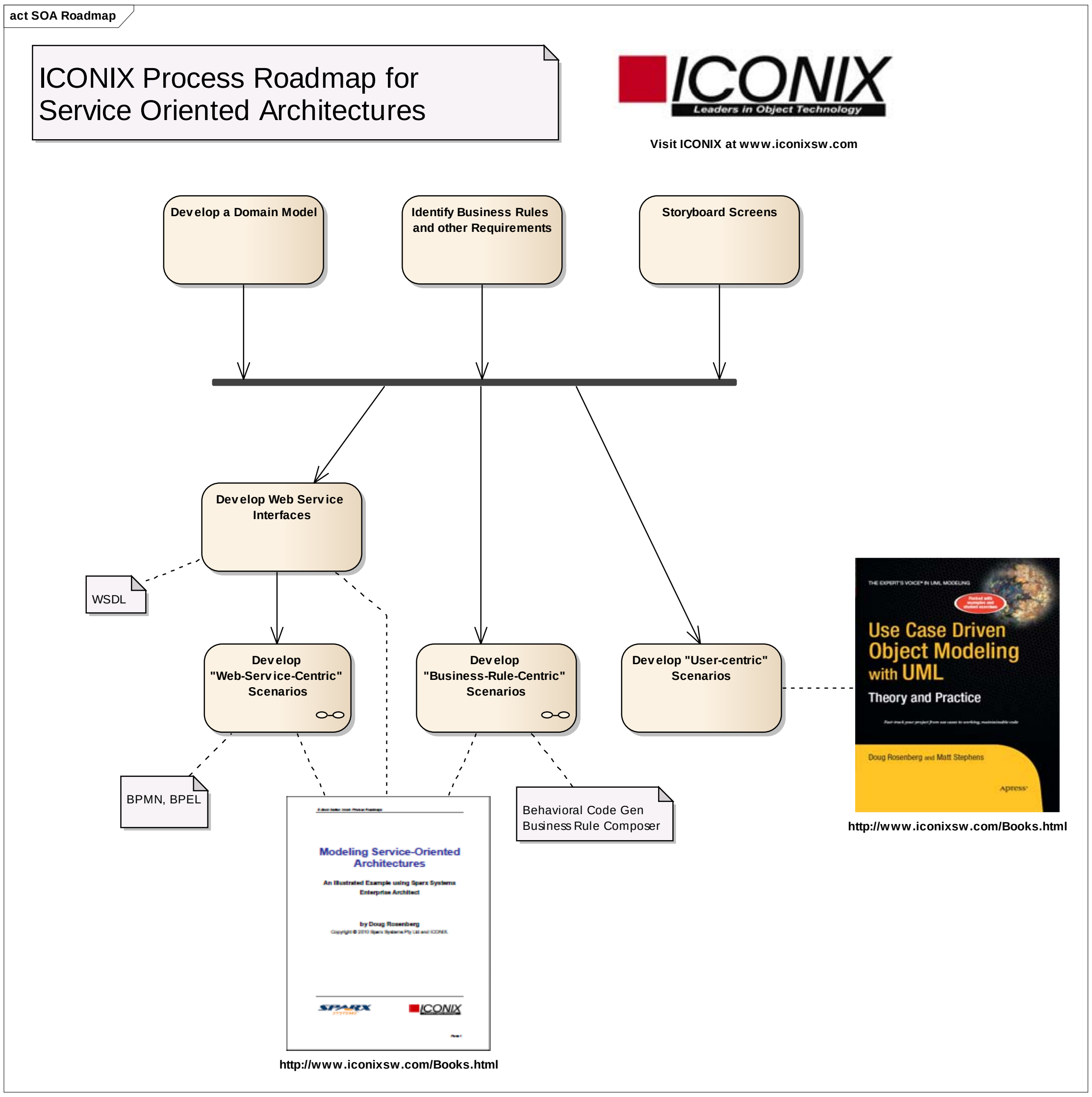

act SOA Roadmap
ICONIX Process Roadmap for Service Oriented Architectures
ICONIX Leaders in Object Technology
Visit ICONIX at www.iconixsw.com
Develop a Domain Model
Identify Business Rules and other Requirements
Storyboard Screens
Develop Web Service Interfaces
WSDL
Develop "Web-Service-Centric" Scenarios
Develop "Business-Rule-Centric" Scenarios
Develop "User-centric" Scenarios
BPMN, BPEL
Behavioral Code Gen
Business Rule Composer
Modeling Service-Oriented Architectures
An Illustrated Example using Sparx Systems Enterprise Architect
by Doug Rosenberg
Use Case Driven Object Modeling with UML
Theory and Practice
Doug Rosenberg and Matt Stephens
http://www.iconixsw.com/Books.html
http://www.iconixsw.com/Books.html

Figure 1-1. ICONIX Process Roadmap for Service-Oriented Architecture Development

[1] These now include roadmaps for Embedded Real-Time Systems, Testing (aka Design Driven Testing), Algorithm Intensive Development, and Business Modeling, in addition to ICONIX Process for Software (the original "Use Case Driven ICONIX Process"). See www.iconixsw.com for more details.

As you can see, the top-level roadmap includes some “common stuff” like figuring out your requirements, and modeling the problem domain (we’ll discuss the “common stuff” in Chapter 2), and then provides multiple paths for developing different flavors of business processes, all the way to code.

The roadmap diagram above reflects the philosophy that when you set out to implement a system using an SOA, there will effectively be a mix of 3 different kinds of scenarios:

1) **Scenarios that use web services** (for which we will develop web service interfaces using WSDL, and orchestrate the use of those web services using BPMN and BPEL.
2) **Scenarios that enforce business rules** (for which we will use activity diagrams and the business rule composer)
3) **Regular old software use cases** (which we won't discuss in this book because we’ve covered it quite thoroughly in my other books)

Most systems will contain a mix of these different types of scenarios.

In many cases (but not all), the *web service scenarios* will cross Business To Business (B2B) boundaries, e.g.: a car rental reservation system talking to a credit card company to run a payment transaction.

The *business rule scenarios* are more likely to be within the boundaries of one business, e.g. the car rental system enforcing eligibility rules on driver age, credit score, etc. There is little or no user interface in these business rule scenarios, so they can be completely code generated from an activity diagram using the business rule composer.

Finally there is the user interface. These would be the screens of the car rental system that the reservations agent accesses, and which would trigger the business rule checks and B2B transactions. These parts of the system are best modeled with use cases.

Where a business process can be implemented via web services, we follow the branch on the left: ***web service interface definition using WSDL, and orchestration using BPMN and BPEL***. We’ll illustrate this with our Car Rental example in Chapters 2 and 3.

- For processes that are primarily focused on enforcing business rules, we take the right branch: ***Behavioral Code Generation from Activity Diagrams using the Business Rule Composer***. We’ll explain how to use the business rule composer and talk about behavioral code generation in Chapter 4.
- And finally, for those user-centric processes that use a GUI, we model them following a *use case driven* approach that’s out of scope for this book, but well documented in my other books[2].
- With all of these branches, we can compile and build using either Eclipse or Visual Studio, and use the Sparx Systems “MDG Integration” technology to keep models and code in-synch. We’ll cover this capability in Chapter 5.

[2] See “Use Case Driven Object Modeling with UML – Theory and Practice” by Doug Rosenberg and Matt Stephens, from Apress

SOA, BPEL, WSDL – Publishing Your Services on the Web

In SOA-based systems, it's possible to implement many business processes by using previously developed "web services". The term "orchestration" is often used to describe implementing a business process by using a number of web services in a collaborative way. In order to better understand this concept, we need to begin explaining some acronyms. We'll give you a brief intro to web services in Chapter 2. For now, we'll start with these two definitions:

***WSDL** (Web Service Definition Language) is an XML-based language that describes web services and how to access them.*

***BPEL** is actually short for Web Service Business Process Execution Language (WS-BPEL). BPEL defines business processes that interact with external entities that are defined using WSDL.*

BPEL is an orchestration language that describes ***messages to and from a business process***. Those messages are defined using WSDL.

Note that it's possible to draw BPMN (Business Process Modeling Notation) diagrams to model business processes without generating BPEL (Business Process Execution Language). And, you can model BPEL using other graphical notations besides BPMN. But one of the more interesting strategies, which we'll explore, involves *using BPMN to model BPEL processes*[3].

Figure 1-2 shows the roadmap basic steps for using BPMN to model BPEL, and generating WSDL.

[3] We found *"Using BPMN to model a BPEL Process"* by Stephen A. White, IBM Corporation, to be a very useful article.

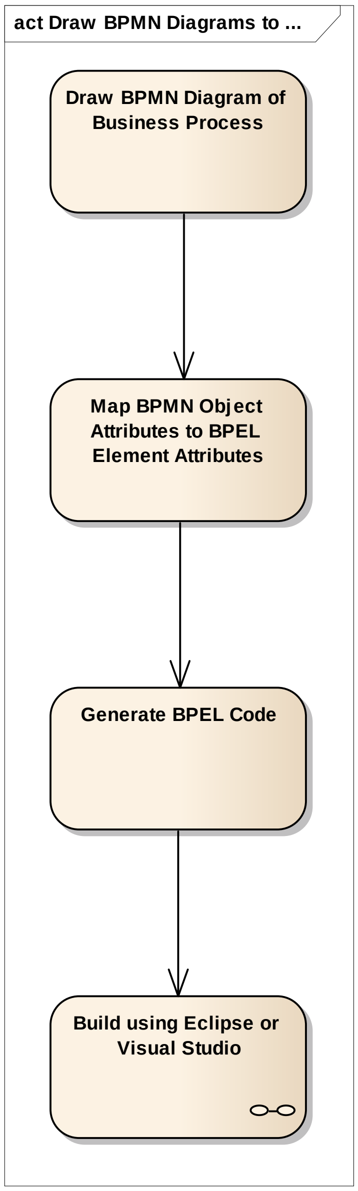

act Draw BPMN Diagrams to ...
Draw BPMN Diagram of Business Process
Map BPMN Object Attributes to BPEL Element Attributes
Generate BPEL Code
Build using Eclipse or Visual Studio

Figure 1-2. We use BPMN to model BPEL, then generate BPEL code

We'll develop web service interfaces using WSDL in Chapter 2, and then further illustrate our roadmap with a BPMN diagram and BPEL/WSDL code for our Car Rental System in Chapter 3.

Business Processes Must Satisfy Business Rules

Even in a service oriented system, not all business processes can be implemented by orchestrating web services. Some business processes will involve user interfaces, and some will be focused on enforcing business rules. We won't discuss GUI-based software use cases in this book, as that process is well documented elsewhere, but we will spend a fair amount of time discussing some new ways to implement those processes that are focused on enforcing business processes.

In Chapter 4 we'll introduce you to the unique capabilities of Enterprise Architect's Business Rule Composer. Starting from the Domain Model and the requirements that we'll present in Chapter 2, we'll take you through the process of creating Activity Diagrams for Business Processes, stereotyping Actions as RuleTasks, and finally, linking the business rules to the rule tasks in preparation for behavioral code generation.

Figure 1-3 shows how it works.

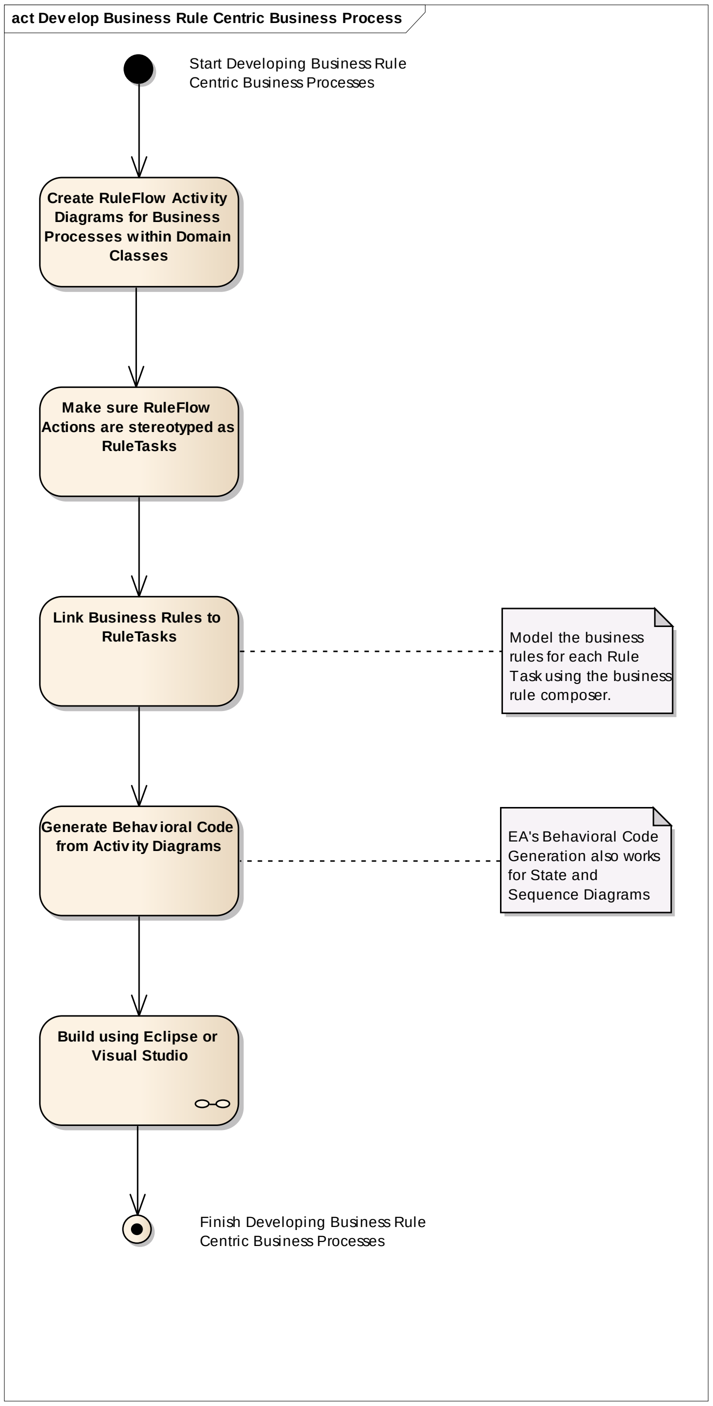

act Develop Business Rule Centric Business Process
Start Developing Business Rule Centric Business Processes
Create RuleFlow Activity Diagrams for Business Processes within Domain Classes
Make sure RuleFlow Actions are stereotyped as RuleTasks
Link Business Rules to RuleTasks
Model the business rules for each Rule Task using the business rule composer.
Generate Behavioral Code from Activity Diagrams
EA's Behavioral Code Generation also works for State and Sequence Diagrams
Build using Eclipse or Visual Studio
Finish Developing Business Rule Centric Business Processes

Figure 1-3. Roadmap for developing "business-rule-centric" scenarios using Enterprise Architect's Business Rule Composer

Once again, we'll illustrate our roadmap with an example of how to use the Rule Composer for our Car Rental System.

Behavioral Code Generation – A Quantum Leap in Tools Capability

One of the common themes of all of our ICONIX Process Roadmaps, is that they all get you to code. We've always believed that processes that only get you halfway to code (or less) are much better in theory than they are in practice (because in practice, programmers tend to ignore them). Enterprise Architect, since its early days, has excelled at ***code engineering*** (forward and reverse engineering for a wide range of languages, powered by customizable code generation templates). But those already strong capabilities have recently taken a quantum leap in power.

The Business and Systems Engineering Edition of Enterprise Architect supports behavioral code generation from activity diagrams, state diagrams and sequence diagrams. Behavioral code generation is a major advancement over generation of "class headers" (which has been the de-facto meaning of "code generation" for more than a decade).

Enterprise Architect uniquely supports generation of complete algorithmic logic, in a variety of languages. When you've used the business rule composer to associate business rules with the rule tasks, you can visually trace requirements all the way to code, since the business rules are propagated into the generated code as comments.

We'll show examples of automatically generated Java code, and automatically generated C# code for our Car Rental System.

Integration with IDEs – Keeping Model and Code Synchronized

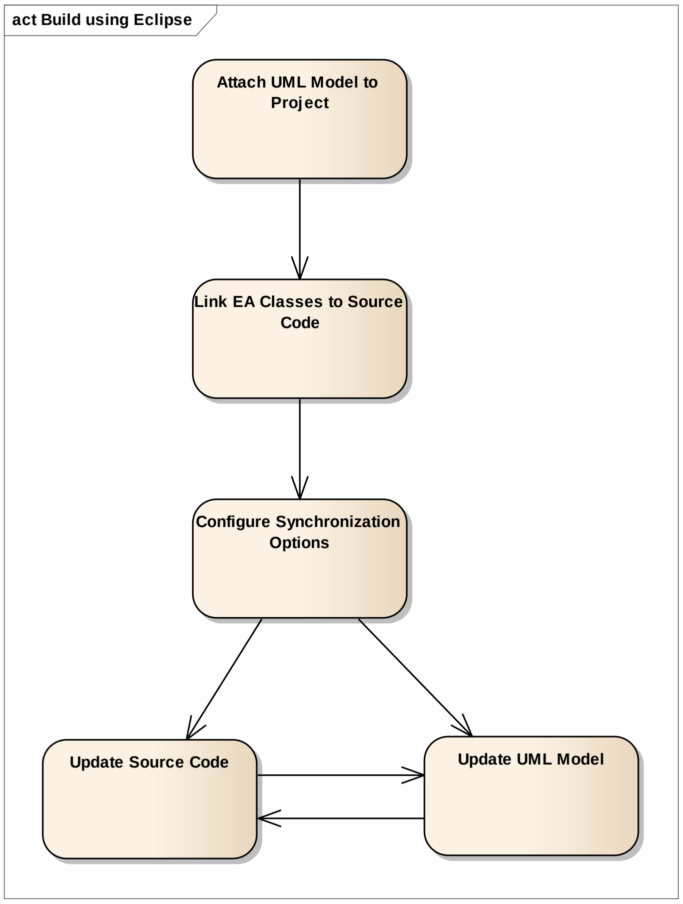

act Build using Eclipse
Attach UML Model to Project
Link EA Classes to Source Code
Configure Synchronization Options
Update Source Code
Update UML Model

Figure 1-4. Licenses for MDG Integration for Visual Studio and Eclipse are included in the Enterprise Architect Business and Software Engineering Edition.

Since well before UML even existed, one of the biggest issues with modeling software has been keeping the model and the source code synchronized. The typical experience *used to be* that UML models were most useful in getting the first release of software completed, but after the initial release, the code would evolve independently in a development environment, and the UML model would rapidly grow obsolete.

With the evolution of agile methodologies, this situation often led to projects abandoning UML modeling entirely, as agile methods specify many frequent releases of the software, and getting to the first release became a smaller and smaller percentage of solving the overall problem.

We'll explain how to beat this problem entirely by using Sparx Systems MDG Integration for Visual Studio and for Eclipse, both of which are included in the Enterprise Architect Business and Software Engineering Edition.

Finally, we'll demonstrate integration with Visual Studio with the C# code for our Car Rental System, and integration with Eclipse using Car Rental Java code. With that, all the steps in our roadmap for SOA development will be complete.

Chapter 2

Getting Organized for SOA

Before undertaking any SOA project (or any other sort of project), it's a good idea to understand some basic fundamentals about what you're planning to build. As Figure 2-1 shows, three major components of this understanding are:

- **Modeling the problem domain**
- **Identifying business rules and other requirements**
- **Storyboarding the user experience (i.e. screens)**

These elements are common to the vast majority of software systems. Once you've developed the foundation of your model, you can choose to develop various operational scenarios using web services, using normal GUI-based use cases, or to enforce business rules. In general, the web service and business rule centric scenarios "hang off" of the software use cases. So the use cases form the glue that holds the model together.

Service-oriented systems will make use of web services. In order to use web services, a system's interfaces must be defined using Web Service Definition Language (WSDL). We'll walk you through this process later in this chapter.

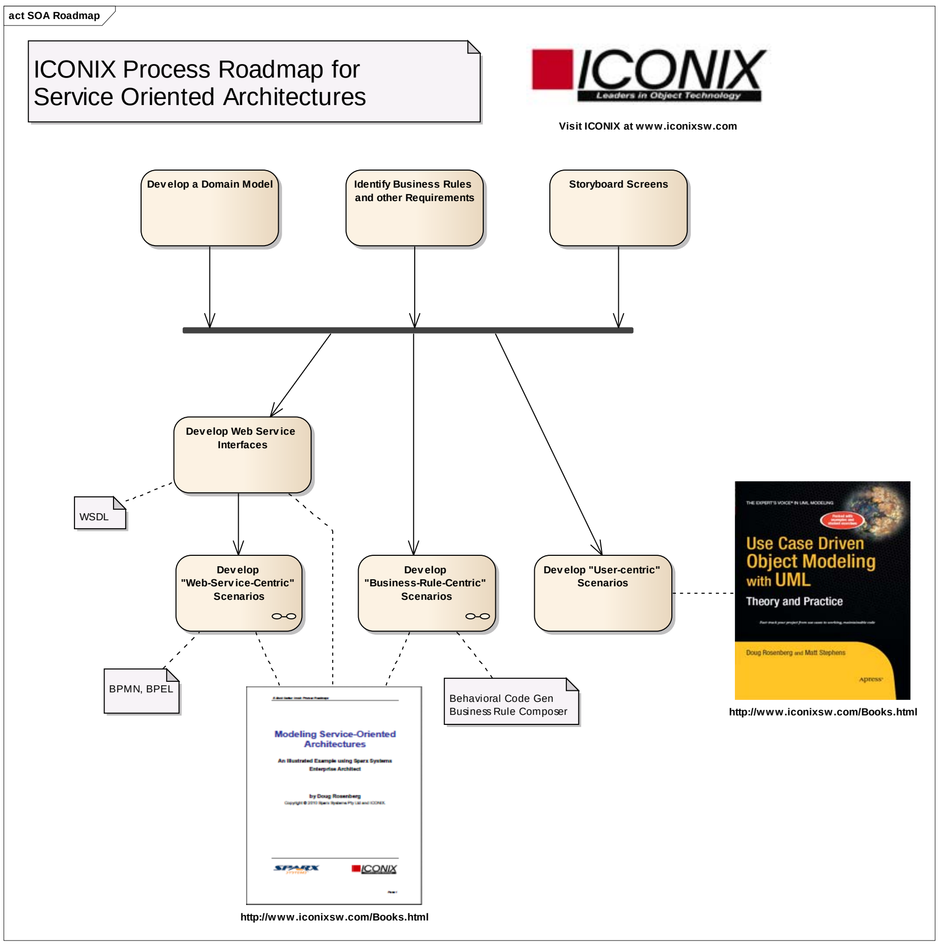
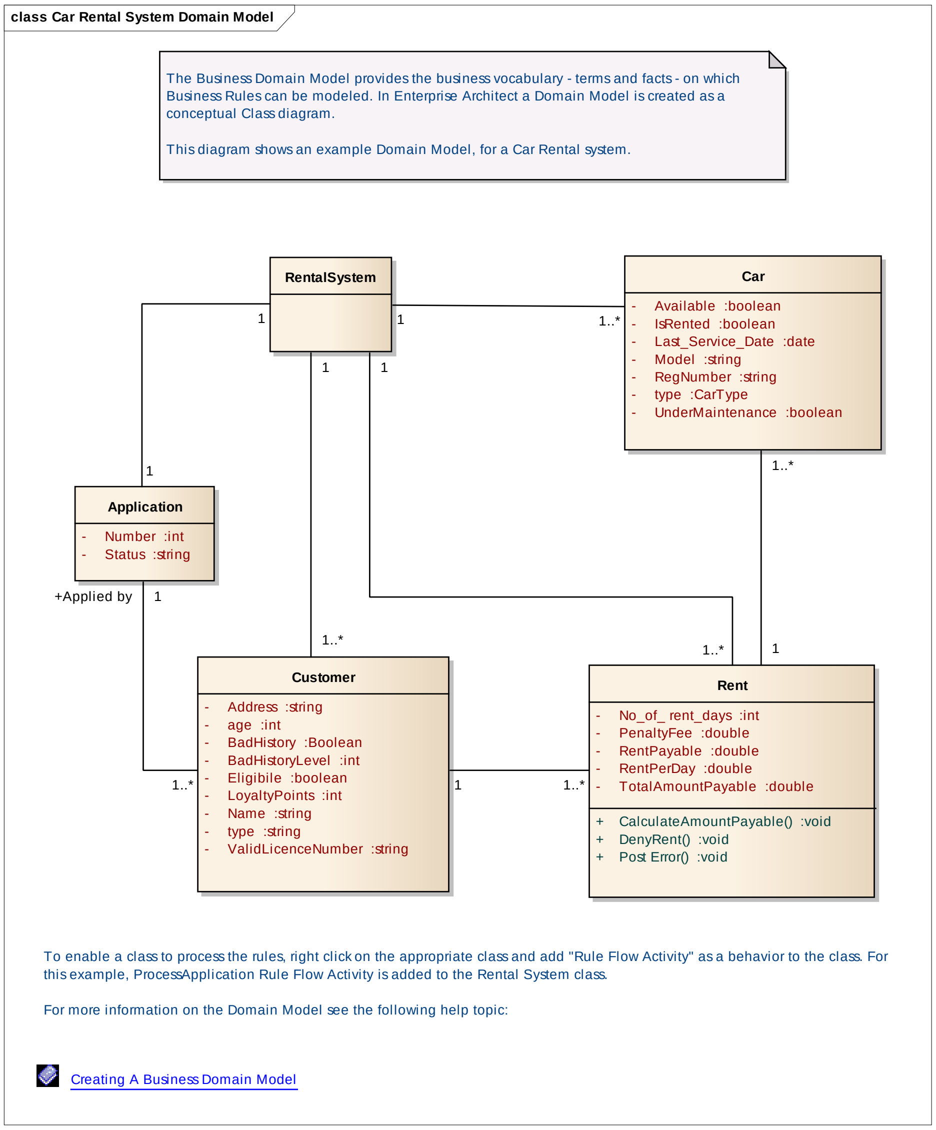

act SOA Roadmap
ICONIX Process Roadmap for Service Oriented Architectures
ICONIX Leaders in Object Technology
Visit ICONIX at www.iconixsw.com
Develop a Domain Model
Identify Business Rules and other Requirements
Storyboard Screens
Develop Web Service Interfaces
WSDL
Develop "Web-Service-Centric" Scenarios
Develop "Business-Rule-Centric" Scenarios
Develop "User-centric" Scenarios
Use Case Driven Object Modeling with UML Theory and Practice
http://www.iconixsw.com/Books.html
BPMN, BPEL
Modeling Service-Oriented Architectures
Behavioral Code Gen
Business Rule Composer
http://www.iconixsw.com/Books.html

Figure 2-1. It's important to develop an understanding of the problem domain, business rules, and user experience early in the project.

Introducing the Car Rental System Example

For the remainder of this book, we'll be illustrating our SOA Roadmap using an example project – a Car Rental System. We'll use this example to illustrate developing a web service-centric scenario using BPMN/BPEL/WSDL, and we'll also use it to demonstrate the capabilities of the Sparx Systems *Business Rule Composer* for behavioral code generation.

We'll begin with a simple domain model (also called a fact model), that shows the main objects in our problem domain. While a complete tutorial on domain modeling is out of the scope of this book, there's a full chapter devoted to this topic in *Use Case Driven Object Modeling*[4]. Figure 2-2 shows the domain model for our Car Rental System.



The Business Domain Model provides the business vocabulary - terms and facts - on which Business Rules can be modeled. In Enterprise Architect a Domain Model is created as a conceptual Class diagram.

This diagram shows an example Domain Model, for a Car Rental system.

To enable a class to process the rules, right click on the appropriate class and add "Rule Flow Activity" as a behavior to the class. For this example, ProcessApplication Rule Flow Activity is added to the Rental System class.

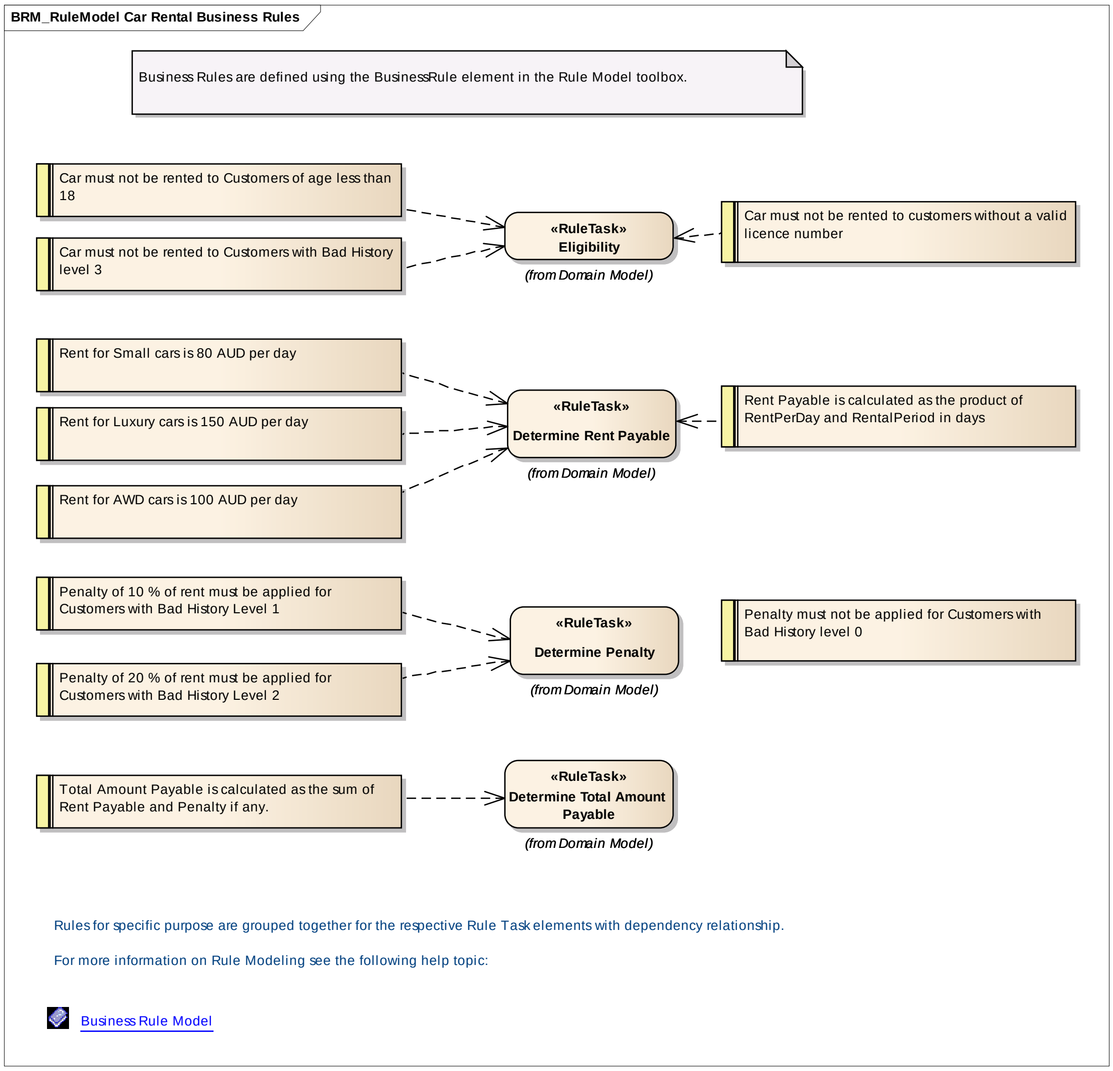
For more information on the Domain Model see the following help topic:

Figure 2-2 Car Rental System Domain (Fact) Model

[4] Use Case Driven Object Modeling with UML – Theory and Practice. Doug Rosenberg and Matt Stephens, Apress.

As you can see, the domain model establishes the vocabulary we use to describe our system, and shows relationships between objects in the problem domain. In most cases these relationships include UML generalization and aggregation relationships (not shown here).

Business Rules are represented in Enterprise Architect as stereotyped **Requirements**[5]. Figure 2-3 shows a Requirement Diagram for our Car Rental System that organizes these business rules according to different `RuleTasks`. We'll talk more about `RuleTasks` in Chapter 4.



BRM_RuleModel Car Rental Business Rules
Business Rules are defined using the BusinessRule element in the Rule Model toolbox.
Car must not be rented to Customers of age less than 18
Car must not be rented to Customers with Bad History level 3
«RuleTask» Eligibility
(from Domain Model)
Car must not be rented to customers without a valid licence number
Rent for Small cars is 80 AUD per day
Rent for Luxury cars is 150 AUD per day
Rent for AWD cars is 100 AUD per day
«RuleTask» Determine Rent Payable
(from Domain Model)
Rent Payable is calculated as the product of RentPerDay and RentalPeriod in days
Penalty of 10 % of rent must be applied for Customers with Bad History Level 1
Penalty of 20 % of rent must be applied for Customers with Bad History Level 2
«RuleTask» Determine Penalty
(from Domain Model)
Penalty must not be applied for Customers with Bad History level 0
Total Amount Payable is calculated as the sum of Rent Payable and Penalty if any.
«RuleTask» Determine Total Amount Payable
(from Domain Model)
Rules for specific purpose are grouped together for the respective Rule Task elements with dependency relationship.
For more information on Rule Modeling see the following help topic:
Business Rule Model

Figure 2-3. Business Rules are one flavor of Requirement, and can be organized by the RuleTask which will be responsible for enforcing the rule.

[5] For a tutorial on how use Enterprise Architect's relationship matrix for requirements traceability, see the Enterprise Architect for Power Users CD from ICONIX (www.iconixsw.com)

As you'll see in Chapter 4, each of these `RuleTasks` will be elaborated on an activity diagram, and Enterprise Architect's **behavioral code generation** capability will allow us to generate complete algorithmic code for these tasks, with no programming, using the Business Rule Composer.

Before we get to the Business Rule Composer, though, we're going to take a trip through "acronym city" in Chapter 3 and explain how to develop web service-centric scenarios using BPMN, BPEL, and WSDL. As a precursor to that discussion, here's a short intro to web services and WSDL, and how to define them using Enterprise Architect.

A Short Intro to Web Services and WSDL

Suppose you wrote an interesting application (maybe a program that automatically generated a horoscope based on your birthday and today's date), and you wanted to publish your application to the Internet so anybody in the world could purchase their horoscope from you. How would you do that? Most likely, you'd do it with a web service.

Web services support distributed, platform-independent development. Using web services, you can publish any application you choose to build over the web. A web service can be written in any language and hosted on any computer that's connected to the Internet.

The concept behind web services isn't new; previously developed similar approaches include OMG CORBA, Microsoft DCOM, and Java/RMI. You can think of a web service as something like an Internet-enabled API.

You'd describe your horoscope web service using Web Service Description Language (WSDL), an XML-based language that describes the public interface to the web service. WSDL tells you only how you can interact with the web service; it says nothing about how the web service works internally.

The internal details of the web service are specified using a "binding". There's a Java binding, which allows you to define local Java implementations that implement web services, and there's also SOAP (Simple Object Access Protocol) which is an XML protocol that operates over standard HTTP to communicate with web services which are on the Internet.

Using SOAP/HTTP, programs connecting to a web service can read the WSDL to determine what operations are available on the server. Any special data types used are embedded in the WSDL file as XML Schema. The program then uses SOAP to call the operations listed in the WSDL.

WSDL defines Services as collections of Network Endpoints, or Ports; a collection of Ports defines a Service. A Port associates a network address with a reusable binding, and a Message is an abstract description of the data that is being exchanged. A WSDL file has an ***abstract*** section that describes Ports and Messages, and a ***concrete*** section that describes specific instances of their usage.

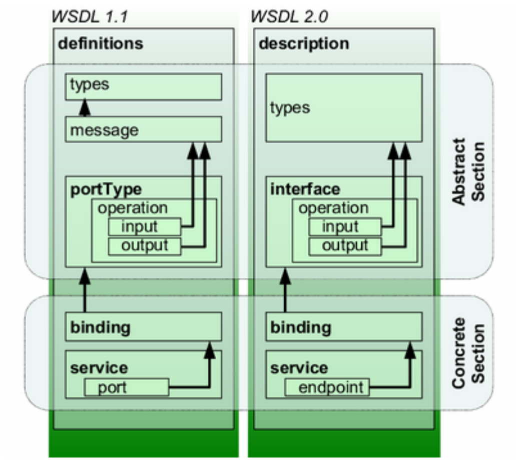


Figure 2-4. WSDL file structure showing Abstract and Concrete sections (from Wikipedia[6])

Developing Web Service Interfaces (WSDL) with Enterprise Architect

Before you can use BPMN and BPEL to orchestrate a collaborating group of web services, you first need to be able to define the web services themselves. The Business and Software Engineering Edition of Enterprise Architect includes a WSDL toolbox, precisely for this purpose. As you can see from Figure 2-5, the elements on this toolbox correspond to the sections in Figure 2-4.

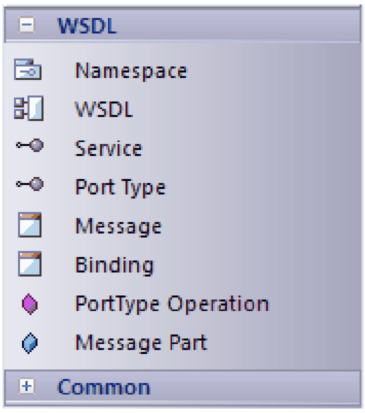


Figure 2-5. Enterprise Architect's Business and Software Engineering Edition supports WSDL development

WSDL Packages in Enterprise Architect are organized into **Types**, **Messages**, **Ports**, **Bindings**, and **Services** as shown in Figure 2-6.

[6] http://en.wikipedia.org/wiki/Web_Services_Description_Language

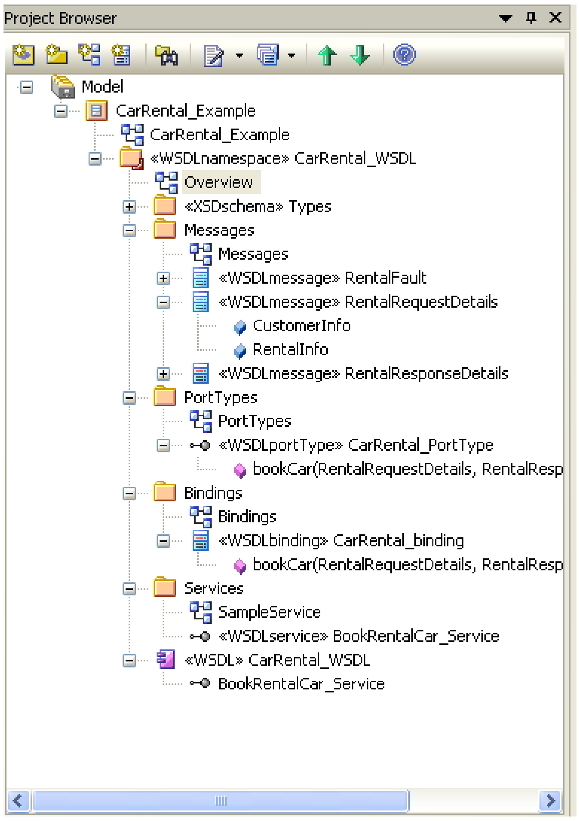

Project Browser
Model
CarRental_Example
CarRental_Example
«WSDLnamespace» CarRental_WSDL
Overview
«XSDschema» Types
Messages
Messages
«WSDLmessage» RentalFault
«WSDLmessage» RentalRequestDetails
CustomerInfo
RentalInfo
«WSDLmessage» RentalResponseDetails
PortTypes
PortTypes
«WSDLportType» CarRental_PortType
bookCar(RentalRequestDetails, RentalResp
Bindings
Bindings
«WSDLbinding» CarRental_binding
bookCar(RentalRequestDetails, RentalResp
Services
SampleService
«WSDLservice» BookRentalCar_Service
«WSDL» CarRental_WSDL
BookRentalCar_Service

Figure 2-6. Organization of a WSDL Package in Enterprise Architect

WSDL documents are represented in Enterprise Architect by UML components stereotyped as WSDL. These components are modeled as direct child elements of the top-level WSDL namespace package. You can create multiple WSDL documents for a single namespace, thus enabling the services for that namespace to be reused and exposed as required across multiple WSDLs.

Figure 2-7 shows a WSDL component, along with the contents of Message, Port, Bindings, and Services Packages, and XSD Schema classes.

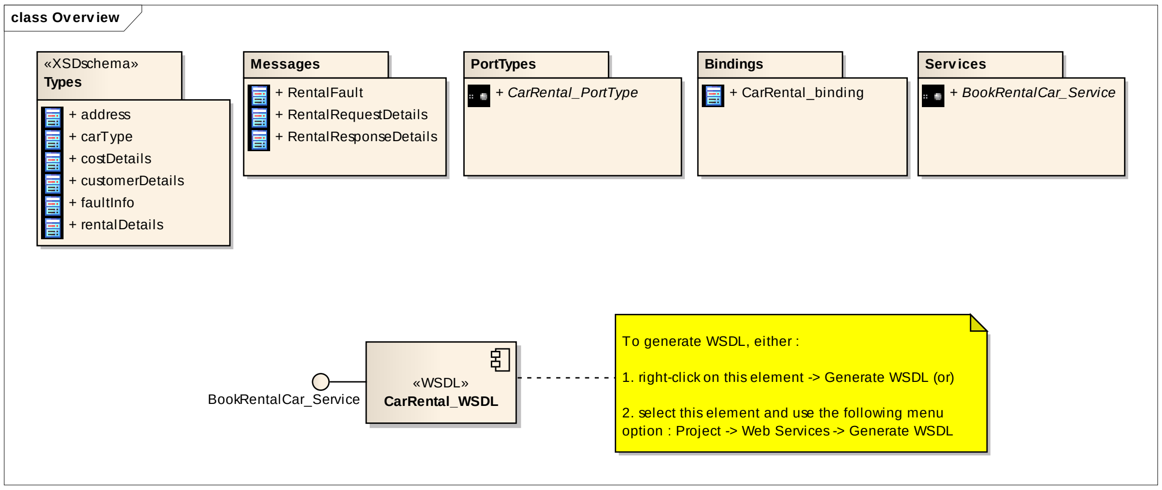


Figure 2-7. WSDL Component for CarRental, exposing the BookRentalCar Service

WSDL Bindings are represented in Enterprise Architect by UML classes stereotyped as `WSDLbinding`. Bindings should be defined under the Bindings package in the WSDL namespace structure. Each `WSDLbinding` class implements the operations specified by a particular WSDL portType interface. Therefore, WSDL Port Types should be defined before creating WSDL bindings.

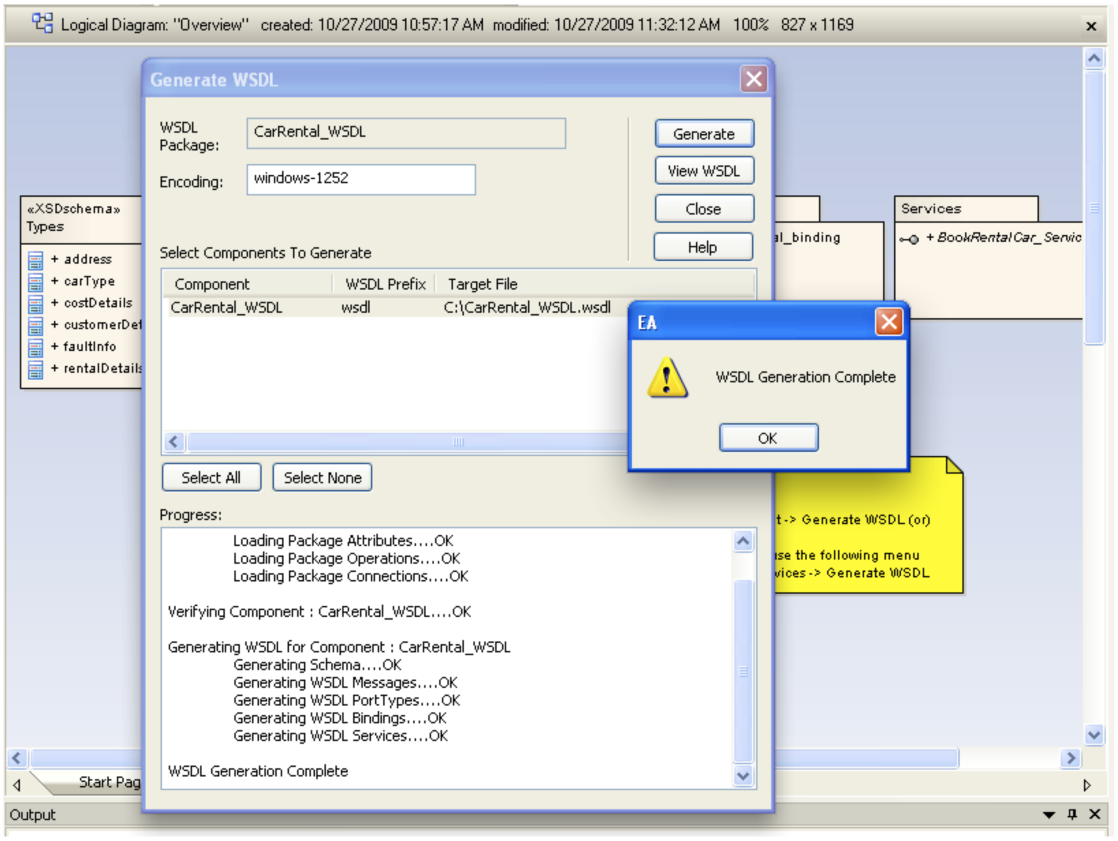
WSDL Port Types are represented in Enterprise Architect by UML interfaces stereotyped as `WSDLportType`. PortTypes should be defined under the PortTypes packages in the WSDL namespace structure. WSDL portType operations are represented in Enterprise Architect by operations defined as part of a `WSDLportType` interface.

WSDL Services are represented in Enterprise Architect by UML interfaces, stereotyped as `WSDLservice`. Services should be defined under the Services packages in the WSDL namespace structure.

WSDL Messages are represented in Enterprise Architect by UML classes stereotyped as `WSDLmessage`. Messages should be defined under the Messages package in the WSDL namespace structure. WSDL message parts are represented in Enterprise Architect by UML attributes defined as part of a `WSDLmessage` class.

Generating WSDL

Once we've defined our Bindings, Port Types, Messages, and Services, it's time to generate WSDL by right-clicking on our WSDL component and choosing Generate WSDL from the context menu. Figure 2-8 shows the WSDL generation in progress.



WSDL Package: CarRental_WSDL

Encoding: windows-1252

Select Components To Generate

Component	WSDL Prefix	Target File
CarRental_WSDL	wsdl	C:\CarRental_WSDL.wsdl

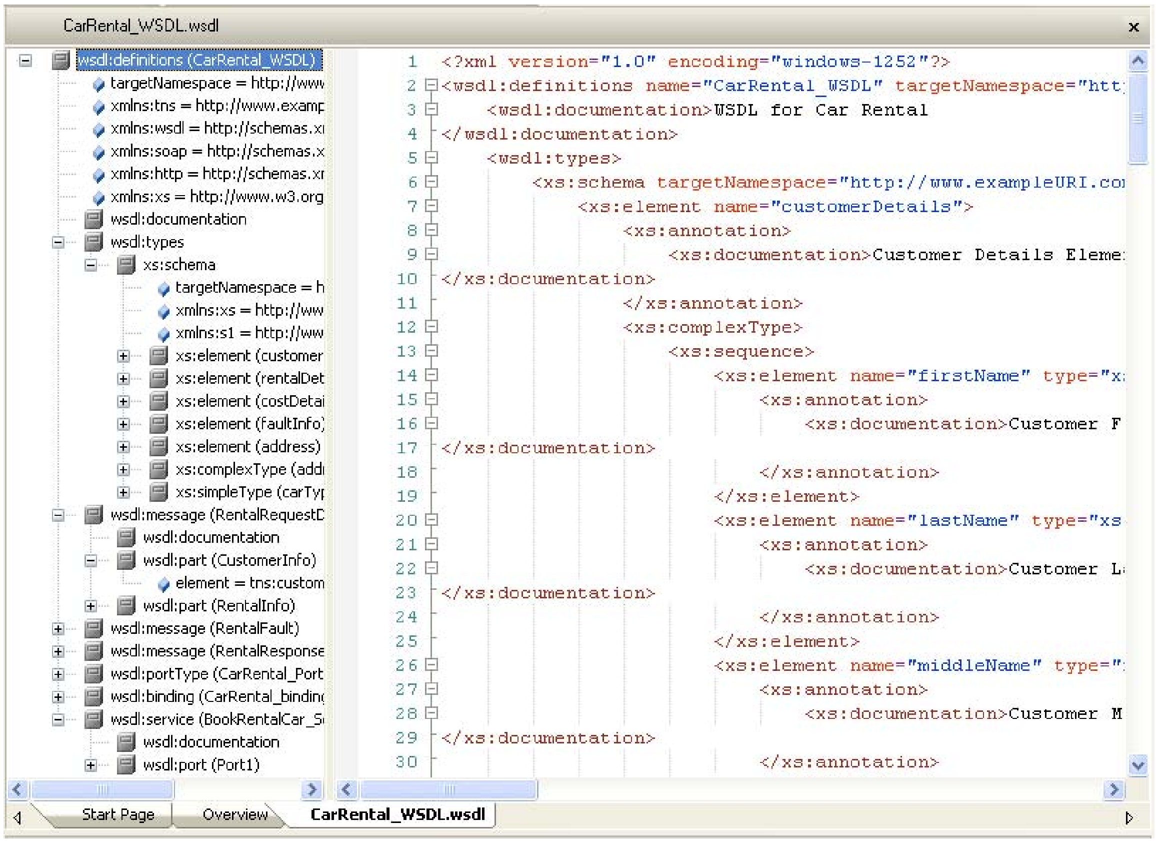
WSDL Generation Complete

Progress:

```
            Loading Package Attributes....OK
            Loading Package Operations....OK
            Loading Package Connections....OK

Verifying Component : CarRental_WSDL....OK

Generating WSDL for Component : CarRental_WSDL
            Generating Schema....OK
            Generating WSDL Messages....OK
            Generating WSDL PortTypes....OK
            Generating WSDL Bindings....OK
            Generating WSDL Services....OK

WSDL Generation Complete
```

Figure 2-8. Enterprise Architect generates WSDL automatically

The result of WSDL generation for our Car Rental component can be seen in Figure 2-9.



```
wsdl:definitions (CarRental_WSDL)
    targetNamespace = http://www
    xmlns:tns = http://www.examp
    xmlns:wsdl = http://schemas.x
    xmlns:soap = http://schemas.x
    xmlns:http = http://schemas.x
    xmlns:xs = http://www.w3.org
    wsdl:documentation
    wsdl:types
        xs:schema
            targetNamespace = h
            xmlns:xs = http://ww
            xmlns:s1 = http://ww
            xs:element (customer
            xs:element (rentalDet
            xs:element (costDetai
            xs:element (faultInfo)
            xs:element (address)
            xs:complexType (addr
            xs:simpleType (carTyp
    wsdl:message (RentalRequestD
        wsdl:documentation
        wsdl:part (CustomerInfo)
            element = tns:custom
        wsdl:part (RentalInfo)
    wsdl:message (RentalFault)
    wsdl:message (RentalResponse
    wsdl:portType (CarRental_Port
    wsdl:binding (CarRental_bindin
    wsdl:service (BookRentalCar_S
        wsdl:documentation
        wsdl:port (Port1)
```

```
<?xml version="1.0" encoding="windows-1252"?>
<wsdl:definitions name="CarRental_WSDL" targetNamespace="htt
    <wsdl:documentation>WSDL for Car Rental
</wsdl:documentation>
    <wsdl:types>
        <xs:schema targetNamespace="http://www.exampleURI.co
            <xs:element name="customerDetails">
                <xs:annotation>
                    <xs:documentation>Customer Details Eleme
</xs:documentation>
                </xs:annotation>
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="firstName" type="x
                            <xs:annotation>
                                <xs:documentation>Customer F
</xs:documentation>
                            </xs:annotation>
                        </xs:element>
                        <xs:element name="lastName" type="xs
                            <xs:annotation>
                                <xs:documentation>Customer L
</xs:documentation>
                            </xs:annotation>
                        </xs:element>
                        <xs:element name="middleName" type="
                            <xs:annotation>
                                <xs:documentation>Customer M
</xs:documentation>
                            </xs:annotation>
```

Figure 2-9. We've successfully generated WSDL for the Car Rental component

We're Ready to Go…

Okay, at this point we've modeled the problem domain, identified our business rules, storyboarded our screens, and defined the interfaces to our Web Services using WSDL. In Chapter 3 we'll walk through the process of orchestrating the web services to do something useful, and in Chapter 4 we'll introduce ***behavioral code generation*** and the Business Rule Composer.

Chapter 3

Orchestrating Web Services with BPMN and BPEL

In this chapter, we'll illustrate a "cookbook" process for developing business processes that use a group of web services collaborating to accomplish their requirements. We'll use the BPEL language to accomplish this. BPEL is an orchestration language that describes ***messages to and from a business process***. Those messages are defined using WSDL.

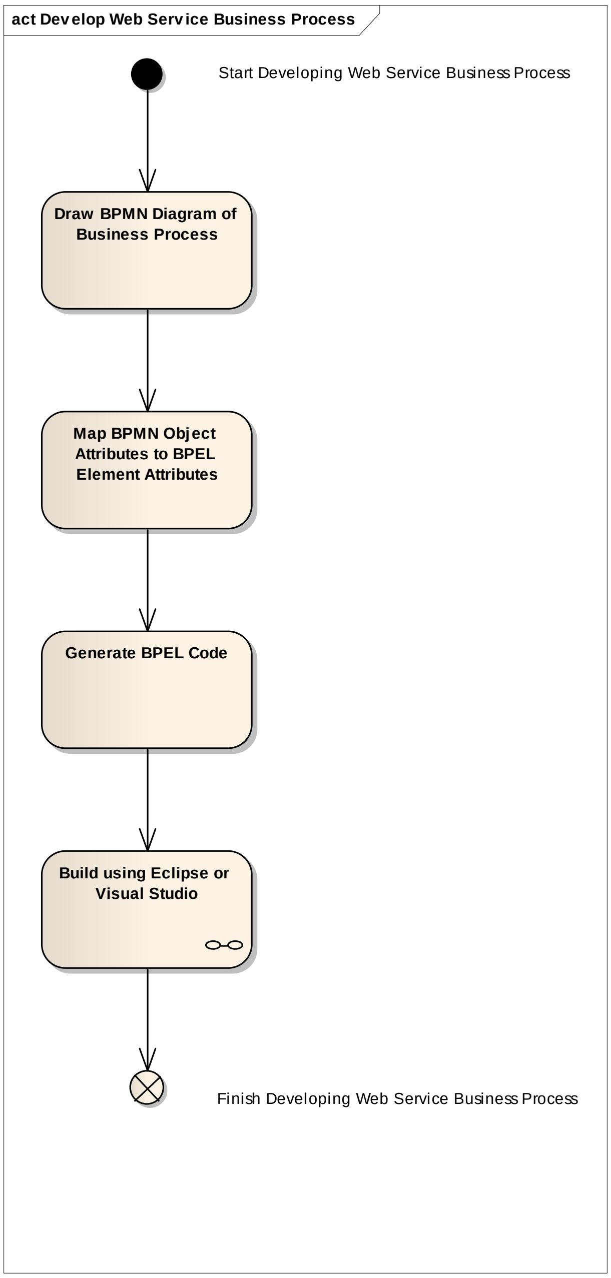

act Develop Web Service Business Process
Start Developing Web Service Business Process
Draw BPMN Diagram of Business Process
Map BPMN Object Attributes to BPEL Element Attributes
Generate BPEL Code
Build using Eclipse or Visual Studio
Finish Developing Web Service Business Process

Figure 3-1. Roadmap for developing web-service centric business processes.

We'll be using the BPMN modeling notation to define our BPEL for the Car Rental system, although other notations could have been used. Let's start with an overview of BPEL.

A Quick Overview of BPEL

BPEL is short for WS-BPEL, which is short for Web Services Business Process Execution Language. You can use BPEL to build web services, to write programs that call web services, and to describe high-level business processes that make use of web services.

BPEL business processes are often used to implement business-to-business (B2B) transactions where one business provides a web service and another business uses it. Our car rental example in this book is an example of this sort of B2B transaction.

As more and more businesses publish functionality to the Internet in the form of web services, the richness of BPEL applications, and therefore the overall importance of BPEL as a development language, will continue to increase.

BPEL is an XML-based programming language. In addition to the logic, which is described in BPEL, data types are defined using XML Schema Definitions (XSD) and input/output is described using WSDL.

BPEL is sometimes referred to as an orchestration language because it supports complex orchestrations (sequences of messages being exchanged) of multiple service applications. Orchestration refers to the central control of the behavior of a distributed system as opposed to choreography, which refers to a distributed system that operates without centralized control. BPEL's orchestration concepts are used by both the external (abstract) and internal (executable) views of a business process.

There is (intentionally) no standard graphical notation for BPEL, and as a result, some vendors have invented their own notations. However, many people use BPMN (Business Process Modeling Notation) as a graphical front-end to capture BPEL process descriptions. We'll be using BPMN to model BPEL in this book with our Car Rental example.

It has been said that BPEL is more popular among web service developers while BPMN is more popular in the business community. Some concepts in BPMN (for example, loops) were left out of BPEL to make the language easier to implement. On the other hand, BPMN only specifies the notation and lacks a complete set of semantics to specify unambiguous code generation. Thus there are some issues in round-trip engineering between BPMN and BPEL.

BPEL in Enterprise Architect

Enterprise Architect currently supports generating BPEL from executable processes. With the help of the BPMN version 1.1 Profile, Enterprise Architect enables you to develop BPEL diagrams quickly and simply. The BPEL facilities are provided in the form of:

- A BPEL Model Template in the Select Models dialog
- A BPEL diagram type, accessed through the New Diagram dialog
- A BPEL Process element in the BPMN 1.1 Core Toolbox pages, which acts as a container from which BPEL can be generated
- Custom dialogs for BPMN elements, highlighting the BPMN Tagged Values relevant to BPEL generation

You can create a BPEL model from the Project Browser, using the Select Model(s) (Model Wizard) dialog.

Enterprise Architect creates a standard package structure for BPEL models, which is shown in Figure 3-2. The standard structure contains the BPEL Process itself and the supporting components (`SupportingElements` and Participant Pools).

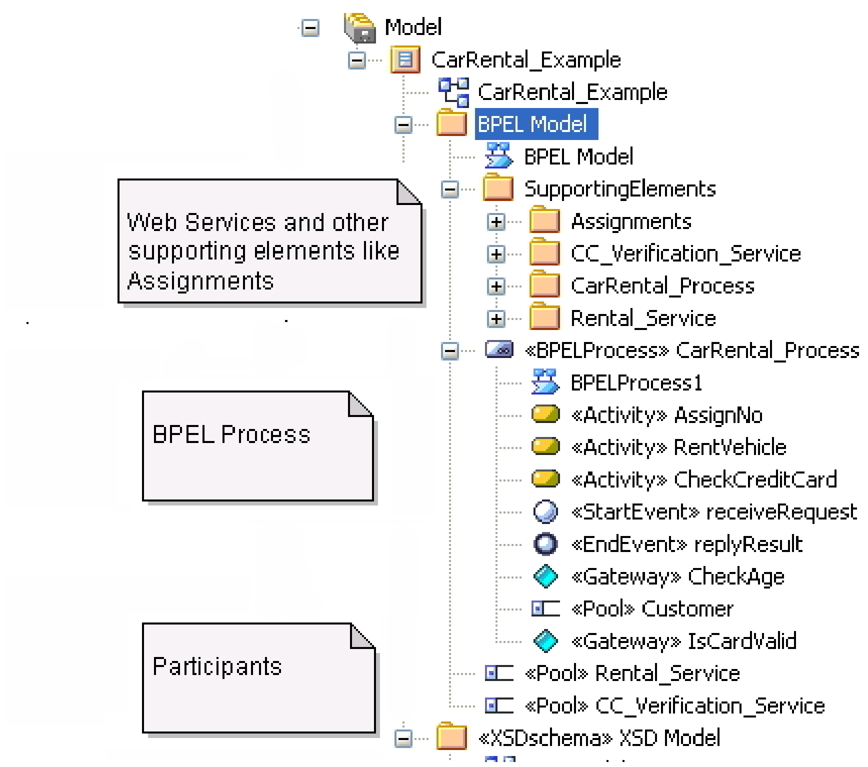


Figure 3-2. Enterprise Architect's standard BPEL Package Structure

Modeling a BPEL Process

The BPEL Process in Enterprise Architect represents the top-level container for the BPEL elements, from which BPEL can be generated. Conceptually it maps to the BPEL process element. BPEL Processes are created using the BPMN 1.1 Toolbox.

The BPEL Process element is a stereotyped Activity that, when created, has a child diagram. That diagram will contain further elements from the BPMN 1.1 Toolbox; specifically: Start Events, End Events, Intermediate Events, Gateways, Activities, Pools, and Notes.

Figure 3-3 shows the BPMN 1.1 Core Toolbox, and we'll discuss each of these elements in our BPMN tutorial, later in this chapter.

Similar to Activity Diagrams, there are 4 main categories of elements on Business Process Diagrams (BPDs): **Flow Objects** such as Events and Activities, **Connecting Objects** such as Messages and Associations, **Swimlanes**, and **other Artifacts** such as Annotations.

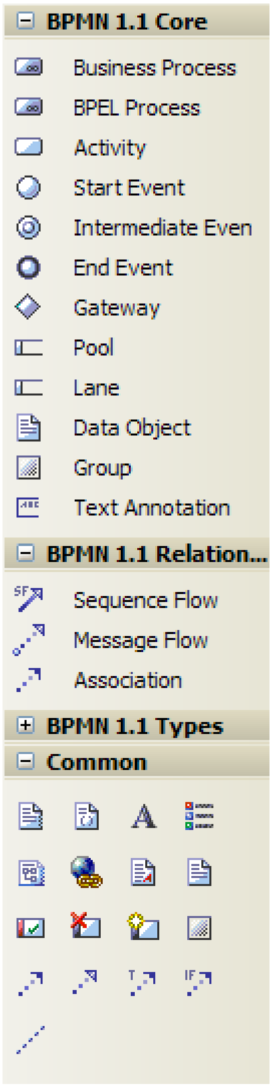


Figure 3-3. Enterprise Architect's BPMN 1.1 Core Toolbox

Note that Pools, Lanes, Data Objects, Groups, and Text Annotations are <u>not</u> mappable to BPEL.

Roadmap: Draw BPMN Diagram of Business Process

Figure 4 shows a BPMN diagram for our BPEL Car Rental Process. The process begins with a Start Event: a Request is received from the Customer. If the customer is of legal age to rent the vehicle, a B2B web service is used to check the Customer's credit card. If the card is valid, another web service is used to rent the vehicle, and a "Success" message is sent. If either the age or credit card checks fail, a "No" message is sent back to the customer.

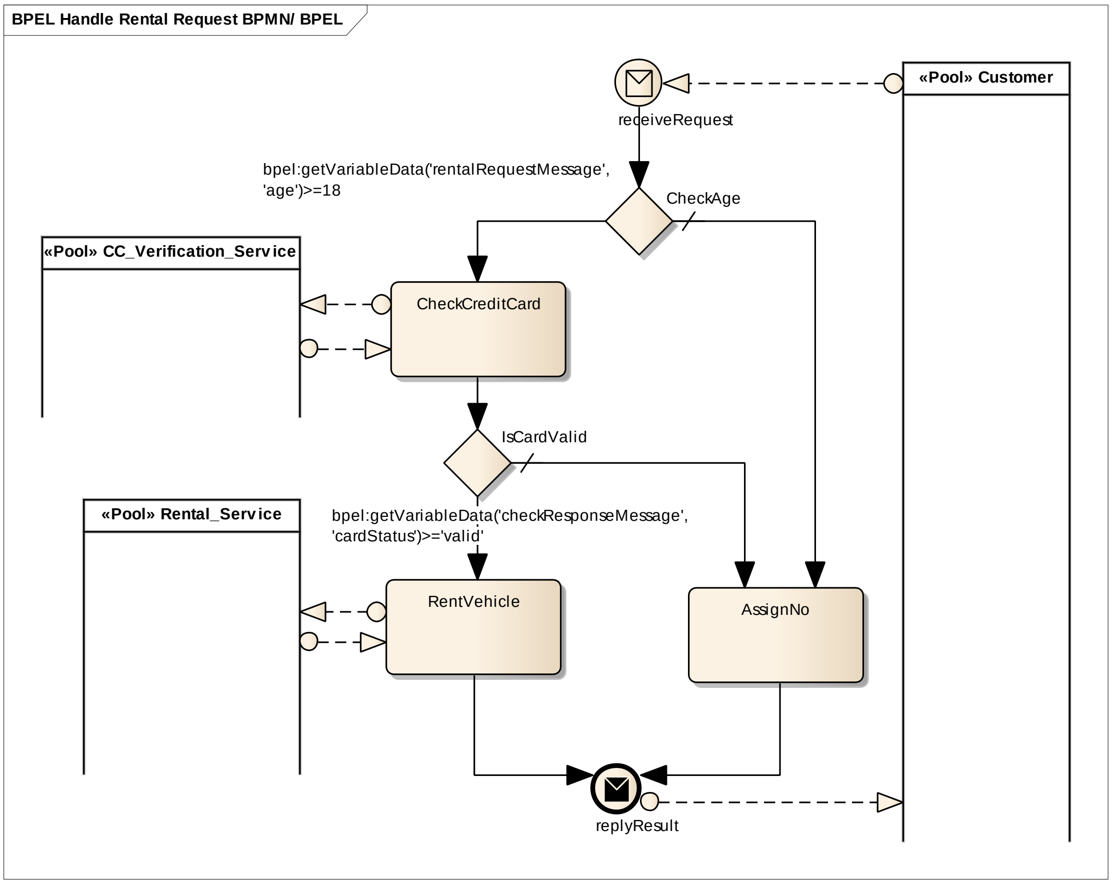

BPEL Handle Rental Request BPMN/ BPEL
«Pool» Customer
receiveRequest
bpel:getVariableData('rentalRequestMessage', 'age')>=18
CheckAge
«Pool» CC_Verification_Service
CheckCreditCard
IsCardValid
«Pool» Rental_Service
bpel:getVariableData('checkResponseMessage', 'cardStatus')>='valid'
RentVehicle
AssignNo
replyResult

Figure 3-4. Orchestrating web services for Car Rental

Everything You Want to Know About BPMN But Were Afraid to Ask

BPMN (Business Process Modeling Notation) provides a graphical notation similar to UML activity diagrams for specifying business processes. BPMN provides a simple, standard notation that is readily understandable by analysts, developers, managers, and end-users.

BPMN is maintained by the OMG, and we'll be referring repeatedly to the "OMG BPMN 1.1 Specification" (aka "the spec") in the next couple of pages.

Start Events and End Events

A **Start Event** indicates where a particular Process begins. Every BPEL Process must begin with a Start Event. A Process can start in several ways, depending on the Trigger Type. The spec defines six types of Trigger (None, Message, Timer, Conditional, Signal, and Multiple). Four of these Trigger types (Message, Timer, Conditional, Multiple) can be mapped to BPEL.

An **End Event** indicates where a particular Process ends. A Process can start in many ways, depending on the Trigger Type, but every BPEL Process must terminate with an End Event. The spec defines eight types of End Event (or Result), which determine the consequence of reaching the End Event. These are: None, Message, Error, Cancel, Compensation, Signal, Terminate, and Multiple. Five of these Result types (Message, Error, Compensation, Terminate, Multiple) can be mapped to BPEL.

Gateways

Gateways control the way in which Sequence Flows converge and diverge within a Process. They provide a gating mechanism that either allows or blocks a Sequence Flow.
The BPMN 1.1 Spec describes four types of Gateways: **Exclusive** (XOR), **Inclusive** (OR), **Complex**, and **Parallel** (AND). Three of these Gateway types (XOR, OR, and AND) can be mapped to BPEL

An **Exclusive Gateway** represents a 'fork in the road'; that is, there can be two or more alternative paths but only one can be taken. Therefore, each path is mutually exclusive (XOR). Exclusive Gateways can be either Data-Based or Event-Based .

Data-Based Exclusive Gateway is the most common type of Exclusive Gateway, where a boolean expression is evaluated to determine the flow path.

With **Event-Based Exclusive Gateways**, the branching is based on the events (such as receiving a message) that occur at that point in the Process, rather than the evaluation of an expression. As an example, when a company receives a response from a customer, they perform one set of activities if the customer responds Yes and another set of activities if the customer responds No. The customer's response determines which path is taken. This Gateway maps to a BPEL Pick element.

With **inclusive gateways**, all the outgoing Sequence Flows with a condition that evaluates to true are taken.

The **parallel gateway** provides a mechanism to create parallel flows.

Pools

A **Pool** represents a Participant in a Process and does not map to any specific BPEL element. Enterprise Architect uses Pools to represent external Participants, with which the BPEL Process communicates. These are 'black box' pools; that is, they are abstract and do not expose any details (they do not contain any BPMN elements inside them).

Activities

An **Activity** represents work that is performed within a Process. An Activity can be modeled as a **Sub-Process** (a compound Activity that is defined as a flow of other BPMN elements) or as a **Task** (an atomic Activity that cannot be broken down into a smaller unit).

Activities - both Tasks and Sub-Processes - can also act as Looping constructs. There are two types of Looping constructs, Standard (while or until) and Multi-Instance (for each). A **Standard Loop** has a boolean Condition that is evaluated after each cycle of the loop. If the evaluation is True, then the loop continues. If Test Time is set to After, the loop is equivalent to a while loop. If Test Time is set to Before, the loop is equivalent to an until loop. A **Multi-Instance Loop** is equivalent to a for each loop and has a numeric expression as a Condition that is evaluated only once before the Activity is performed. The result of the evaluation specifies the number of times the loop is repeated.

The BPMN Specification defines three types of Sub-Process: **Embedded**, **References**, and **Reusable**. Embedded and References Sub-Process types can be mapped to BPEL.

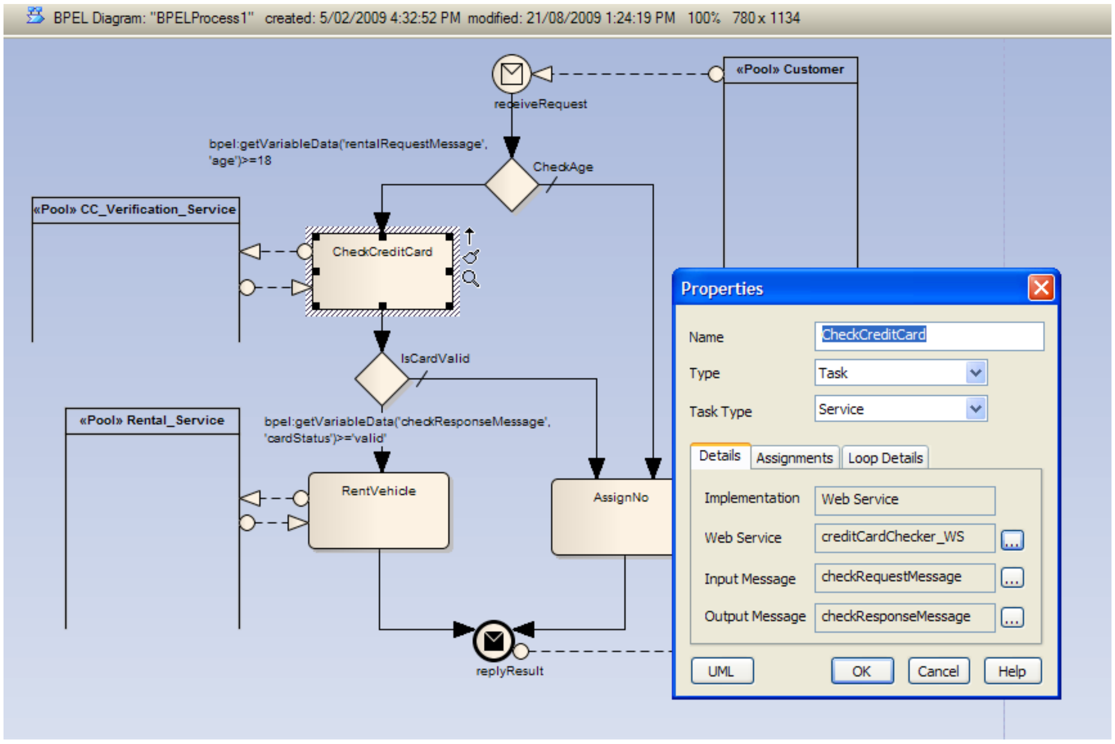
Assignments

A BPMN **Assignment** element enables data to be copied between messages, and new data to be inserted, using expressions within a BPEL Process. A BPMN Assignment element maps to a BPEL assign activity and copies the specified value from the source to the target.

In Enterprise Architect, Assignment elements should be created in the Assignments package in SupportingElements. If they are created elsewhere, they cannot be enacted correctly.

Roadmap: Map BPMN Object Attributes to BPEL Element Attributes

Once we've defined our activities, gateways, and events on our BPMN diagram, we can specify additional BPEL details as attributes on our BPMN elements. In Figure 3-5, we're defining that the CheckCreditCard activity will be implemented as a web service, and will take a `request` message and generate a `result` response.



BPEL Diagram: "BPELProcess1" created: 5/02/2009 4:32:52 PM modified: 21/08/2009 1:24:19 PM 100% 780 x 1134

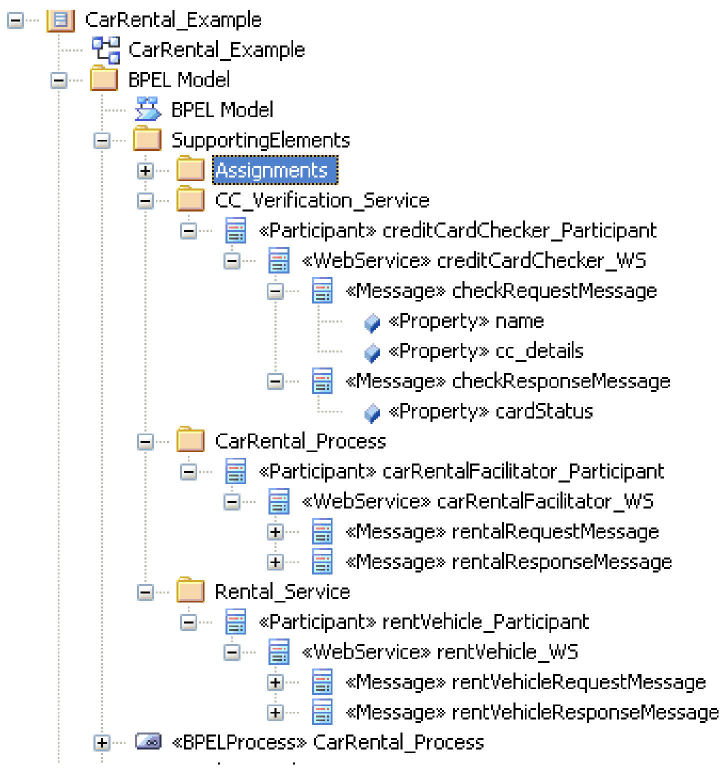
```
bpel:getVariableData('rentalRequestMessage',
'age')>=18
```

```
bpel:getVariableData('checkResponseMessage',
'cardStatus')>='valid'
```

Name	CheckCreditCard
Type	Task
Task Type	Service
Implementation	Web Service
Web Service	creditCardChecker_WS
Input Message	checkRequestMessage
Output Message	checkResponseMessage

Figure 3-5. Defining Messages to and from the CheckCreditCard Web Service

In Figure 3-6, you can see that the `checkRequestMessage` belongs to the `creditCardChecker` web service, and has attributes of `name` and `cc_details`, and the `checkResponseMessage` has an attribute of `cardStatus`.

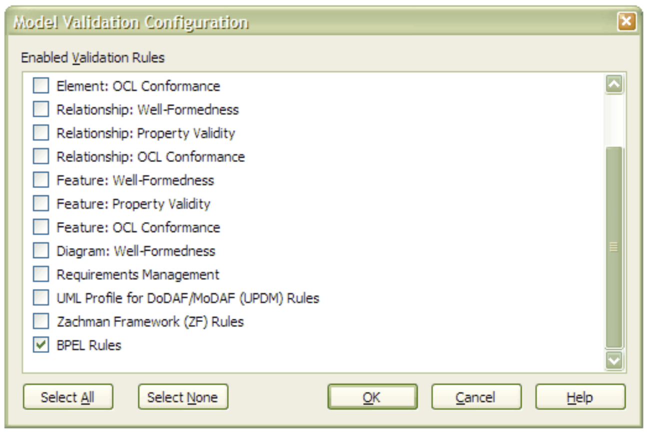


```
⊟ CarRental_Example
    CarRental_Example
  ⊟ BPEL Model
      BPEL Model
    ⊟ SupportingElements
      ⊞ Assignments
      ⊟ CC_Verification_Service
        ⊟ «Participant» creditCardChecker_Participant
          ⊟ «WebService» creditCardChecker_WS
            ⊟ «Message» checkRequestMessage
                «Property» name
                «Property» cc_details
            ⊟ «Message» checkResponseMessage
                «Property» cardStatus
      ⊟ CarRental_Process
        ⊟ «Participant» carRentalFacilitator_Participant
          ⊟ «WebService» carRentalFacilitator_WS
            ⊞ «Message» rentalRequestMessage
            ⊞ «Message» rentalResponseMessage
      ⊟ Rental_Service
        ⊟ «Participant» rentVehicle_Participant
          ⊟ «WebService» rentVehicle_WS
            ⊞ «Message» rentVehicleRequestMessage
            ⊞ «Message» rentVehicleResponseMessage
    ⊞ «BPELProcess» CarRental_Process
```

Figure 3-6. Messages are created inside the Web Service in the Project Browser

BPEL Model Validation

You can use Enterprise Architect's ***Model Validation*** facility to check the validity of the BPEL model (see Figure 3-7). You can validate an entire BPEL Process or a single BPMN element. Note that Enterprise Architect checks for both the UML and the BPEL rules by default. To enable only BPEL rule validation, select only the BPEL Rules checkbox in the Model Validation Configuration dialog.



Model Validation Configuration

Enabled Validation Rules

- [ ] Element: OCL Conformance
- [ ] Relationship: Well-Formedness
- [ ] Relationship: Property Validity
- [ ] Relationship: OCL Conformance
- [ ] Feature: Well-Formedness
- [ ] Feature: Property Validity
- [ ] Feature: OCL Conformance
- [ ] Diagram: Well-Formedness
- [ ] Requirements Management
- [ ] UML Profile for DoDAF/MoDAF (UPDM) Rules
- [ ] Zachman Framework (ZF) Rules
- [✓] BPEL Rules

Figure 3-7. Enterprise Architect validates BPEL Rules

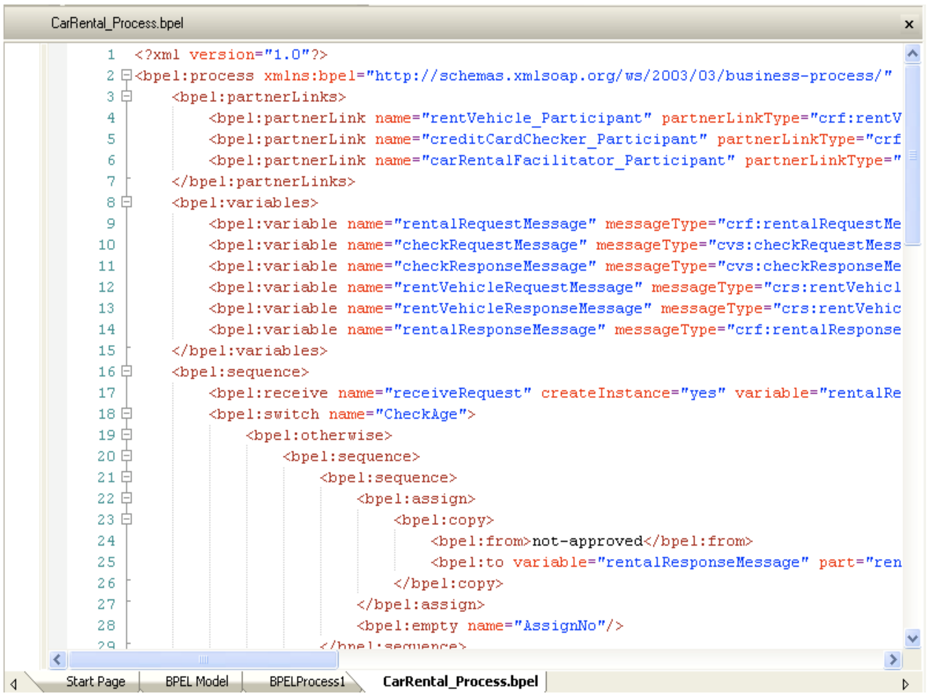
Modeling Restrictions

Following these rules will help your BPEL modeling effort with Enterprise Architect to be more successful:

- Use the elements from the BPMN 1.1 Toolbox pages for BPEL modeling.
- Every BPEL Process and Sub-Process should start with a `StartEvent` and end with an `EndEvent`.
- A `StartEvent` or an `EndEvent` should not be attached to the boundary of a Sub-Process.
- *SequenceFlow Looping* is not supported - only Activity looping is supported. All `SequenceFlows` should flow downstream and not upstream.
- Mapping of an `IntermediateEvent` with multiple triggers to BPEL is not supported.
- Mapping of multi-instance parallel `While` loops to BPEL is not supported.
- Mapping of Independent sub-processes to BPEL is not supported.

Generate BPEL Code

Finally, after you've specified your BPEL Process and validated the model, Enterprise Architect will generate the BPEL code to accomplish the business process (see Figure 3-8).



```
<?xml version="1.0"?>
<bpel:process xmlns:bpel="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
    <bpel:partnerLinks>
        <bpel:partnerLink name="rentVehicle_Participant" partnerLinkType="crf:rentV
        <bpel:partnerLink name="creditCardChecker_Participant" partnerLinkType="crf
        <bpel:partnerLink name="carRentalFacilitator_Participant" partnerLinkType="
    </bpel:partnerLinks>
    <bpel:variables>
        <bpel:variable name="rentalRequestMessage" messageType="crf:rentalRequestMe
        <bpel:variable name="checkRequestMessage" messageType="cvs:checkRequestMess
        <bpel:variable name="checkResponseMessage" messageType="cvs:checkResponseMe
        <bpel:variable name="rentVehicleRequestMessage" messageType="crs:rentVehicl
        <bpel:variable name="rentVehicleResponseMessage" messageType="crs:rentVehic
        <bpel:variable name="rentalResponseMessage" messageType="crf:rentalResponse
    </bpel:variables>
    <bpel:sequence>
        <bpel:receive name="receiveRequest" createInstance="yes" variable="rentalRe
        <bpel:switch name="CheckAge">
            <bpel:otherwise>
                <bpel:sequence>
                    <bpel:sequence>
                        <bpel:assign>
                            <bpel:copy>
                                <bpel:from>not-approved</bpel:from>
                                <bpel:to variable="rentalResponseMessage" part="ren
                            </bpel:copy>
                        </bpel:assign>
                        <bpel:empty name="AssignNo"/>
                    </bpel:sequence>
```

Figure 3-8. Enterprise Architect generates BPEL code from BPMN models

Once the code is generated, you're ready to move into Eclipse or Visual Studio and use the MDG Integration capability of Enterprise Architect to keep your model and the source code tightly linked together. But first, let's take a look at the Business Rule Composer and learn about ***behavioral code generation***.

Chapter 4

Behavioral Code Generation for Business Rules

In this chapter we'll introduce you to some unique capabilities of Enterprise Architect's Business and Software Engineering Edition that provide an entirely new approach to the enforcement of business rules. Continuing our Car Rental System example, we'll show how to use Activity Diagrams to drive behavioral code generation for custom software using the Sparx Systems Business Rule Composer. Figure 4-1 shows our roadmap for "business rule centric" processes.

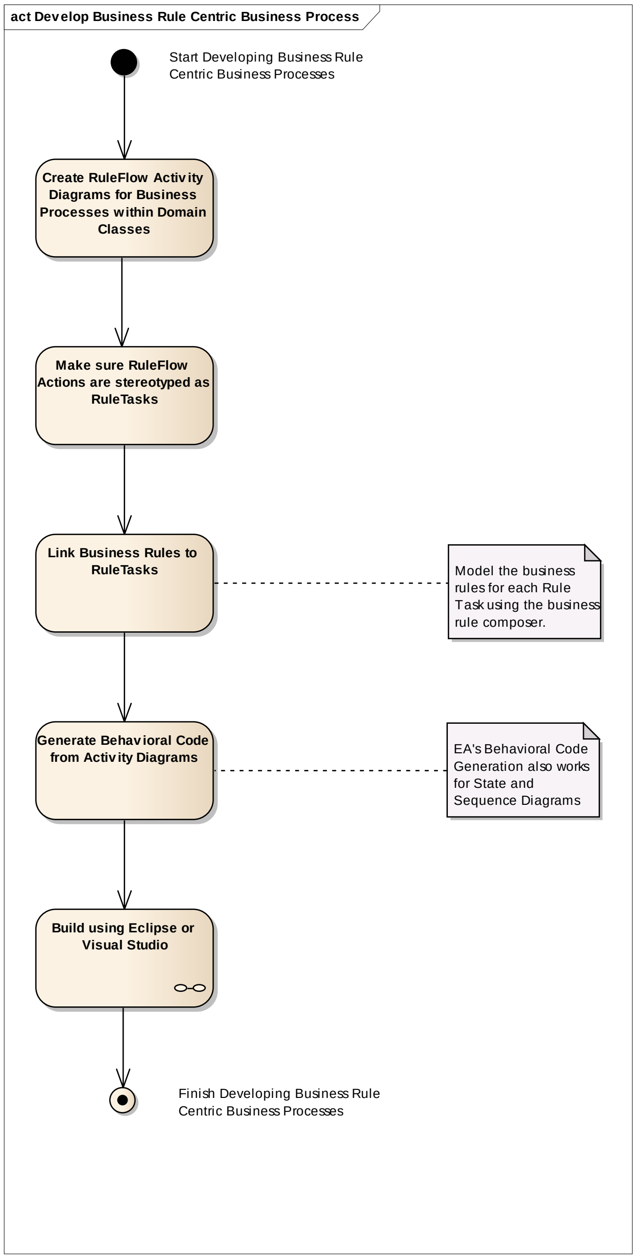

act Develop Business Rule Centric Business Process
Start Developing Business Rule Centric Business Processes
Create RuleFlow Activity Diagrams for Business Processes within Domain Classes
Make sure RuleFlow Actions are stereotyped as RuleTasks
Link Business Rules to RuleTasks
Model the business rules for each Rule Task using the business rule composer.
Generate Behavioral Code from Activity Diagrams
EA's Behavioral Code Generation also works for State and Sequence Diagrams
Build using Eclipse or Visual Studio
Finish Developing Business Rule Centric Business Processes

Figure 4-1. Use Activity Diagrams to model "business-rule centric" processes.

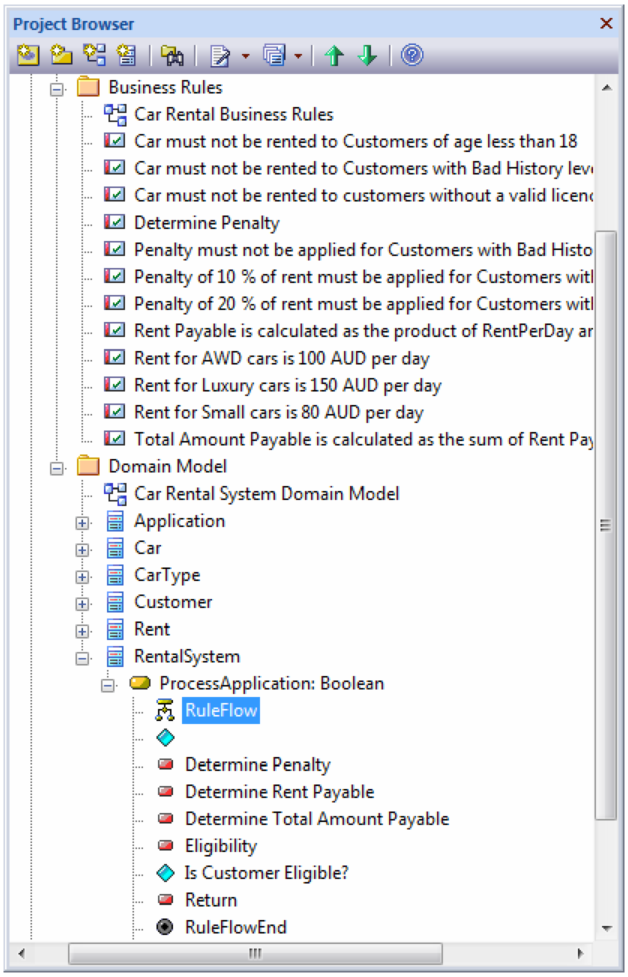
Behavioral Code Generation Includes Logic, Not Just Class Headers

For more than 20 years now, modeling tools have "generated code" from graphical models. Traditionally, this form of code generation has involved taking class definitions and generating headers, with UML attributes and operations turned into data members and function members. With the introduction of Behavioral Code Generation (which works from Activity, State, and Sequence diagrams), Sparx Systems has opened a new chapter on this technology. **It's now possible to generate complete algorithmic logic from the model**.

While Enterprise Architect's behavioral code generator works with State and Sequence diagrams, the example in this chapter focuses on generating code from Activity Diagrams, specifically to implement business rules. The code generation shown in this chapter does not involve a scenario that includes a GUI – it's "pure" algorithmic code.

Roadmap: Create Activity Diagrams for Business Processes, within Domain Classes; Stereotype Actions as Rule Tasks

Let's look at how to process an application for our Car Rental System. The first thing to do is create a RuleFlow diagram – in this case `ProcessApplication`. We'll create this diagram inside of a class in the Domain Model, just as if we were creating an Operation on that class, since that is, in effect, what we're doing (see Figure 4-2).

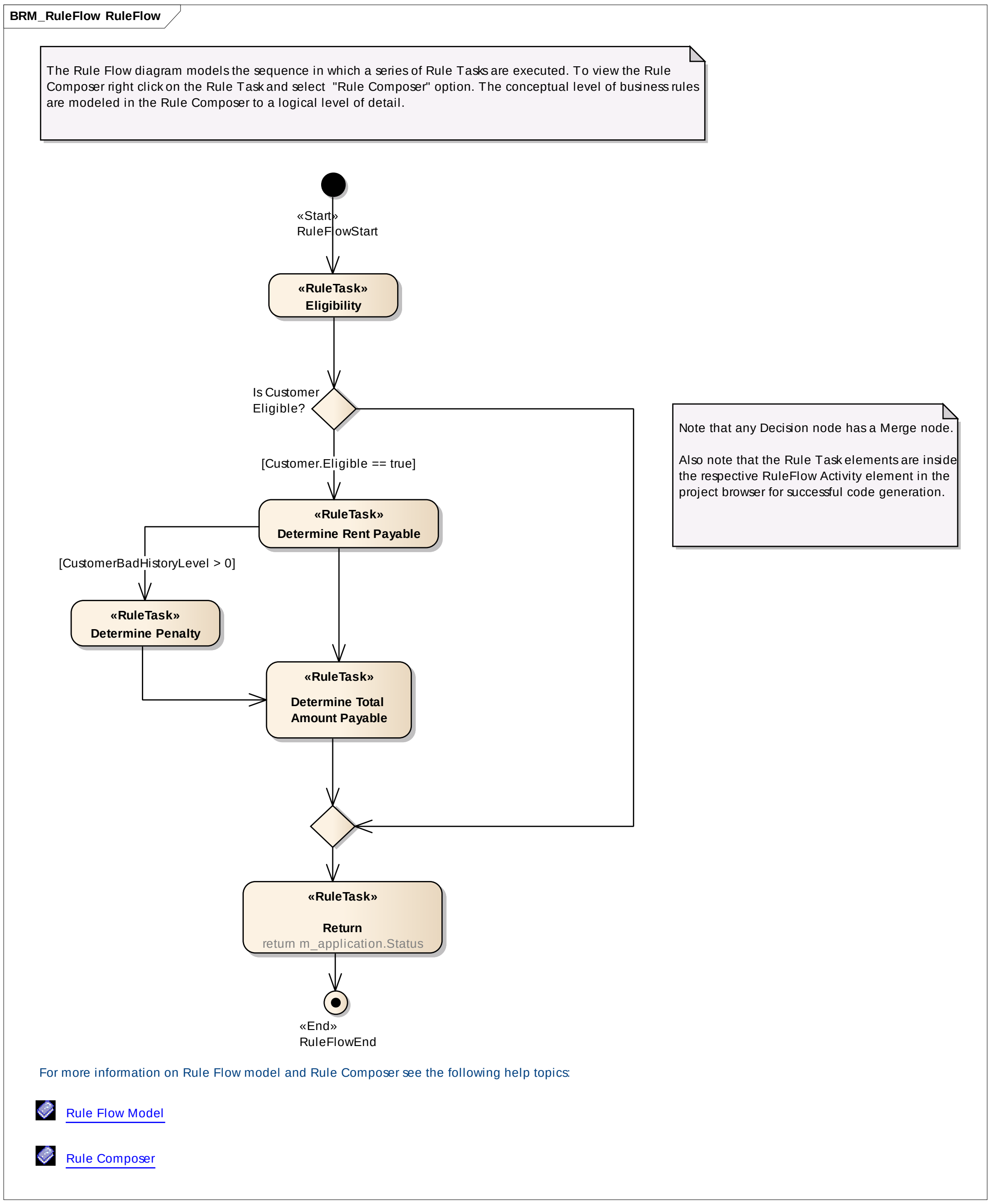


Project Browser

- Business Rules
  - Car Rental Business Rules
  - Car must not be rented to Customers of age less than 18
  - Car must not be rented to Customers with Bad History lev
  - Car must not be rented to customers without a valid licen
  - Determine Penalty
  - Penalty must not be applied for Customers with Bad Histo
  - Penalty of 10 % of rent must be applied for Customers wit
  - Penalty of 20 % of rent must be applied for Customers wit
  - Rent Payable is calculated as the product of RentPerDay ar
  - Rent for AWD cars is 100 AUD per day
  - Rent for Luxury cars is 150 AUD per day
  - Rent for Small cars is 80 AUD per day
  - Total Amount Payable is calculated as the sum of Rent Pa
- Domain Model
  - Car Rental System Domain Model
  - Application
  - Car
  - CarType
  - Customer
  - Rent
  - RentalSystem
    - ProcessApplication: Boolean
      - RuleFlow
      - Determine Penalty
      - Determine Rent Payable
      - Determine Total Amount Payable
      - Eligibility
      - Is Customer Eligible?
      - Return
      - RuleFlowEnd

Figure 4-2. RuleFlow diagrams are created inside Domain Model Classes.

Once we've created the RuleFlow diagram, we'll populate it with Actions, Decisions, etc. We'll stereotype these Actions as <RuleTask>s. If you recall from Chapter 2, we organized our Business Rules by RuleTask. Figure 4-3 shows the result.



The Rule Flow diagram models the sequence in which a series of Rule Tasks are executed. To view the Rule Composer right click on the Rule Task and select "Rule Composer" option. The conceptual level of business rules are modeled in the Rule Composer to a logical level of detail.

Note that any Decision node has a Merge node.

Also note that the Rule Task elements are inside the respective RuleFlow Activity element in the project browser for successful code generation.

For more information on Rule Flow model and Rule Composer see the following help topics:

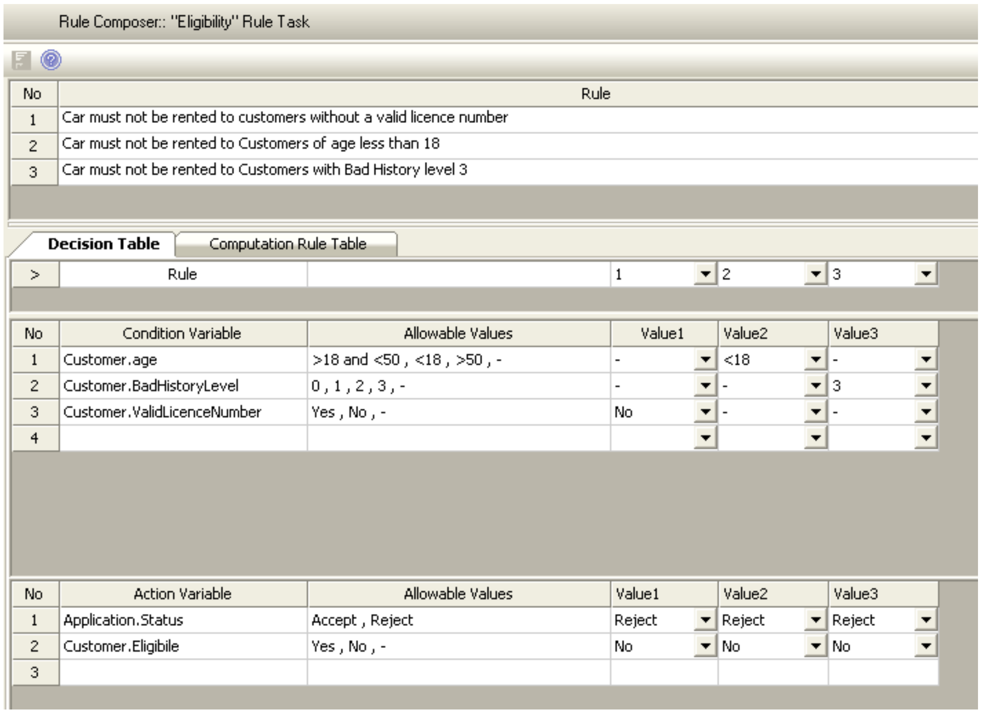
Figure 4-3. RuleFlow diagrams contain Actions stereotyped as RuleTask.

Our algorithm first determines whether the `Customer` is `Eligible` to rent the vehicle, then determines the price, including any possible penalties due to the `Customer`'s history, and returns, presenting a pass/fail application status.

Roadmap: Link Business Rules to RuleTasks

Next, we should specify the conditional logic for each of these RuleTasks, while associating each RuleTask with the specific Business Rules that we're enforcing. As you might have guessed by now, that's where the Business Rule Composer comes into play. Once we've completely specified each RuleTask, we'd like Enterprise Architect to generate 100% complete code for the entire RuleFlow (Activity Diagram). Using Behavioral Code Generation, that's exactly what we'll do.

Figure 4-4 shows the Business Rule Composer for the `Eligibility` RuleTask.


Rule Composer:: "Eligibility" Rule Task

No	Rule
1	Car must not be rented to customers without a valid licence number
2	Car must not be rented to Customers of age less than 18
3	Car must not be rented to Customers with Bad History level 3

Decision Table | Computation Rule Table

>	Rule		1	2	3

No	Condition Variable	Allowable Values	Value1	Value2	Value3
1	Customer.age	>18 and <50 , <18 , >50 , -	-	<18	-
2	Customer.BadHistoryLevel	0 , 1 , 2 , 3 , -	-	-	3
3	Customer.ValidLicenceNumber	Yes , No , -	No	-	-
4					

No	Action Variable	Allowable Values	Value1	Value2	Value3
1	Application.Status	Accept , Reject	Reject	Reject	Reject
2	Customer.Eligibile	Yes , No , -	No	No	No
3					

Figure 4-4. Business Rule Composer showing Rules and Logic for Eligibility.

There are 2 sections on the Rule Composer screen. The top panel shows the Business Rules (from Chapter 2) that we're satisfying within this `Eligibility` RuleTask; the lower section (in this case a Decision Table) has 3 parts: a Condition section to model condition variables, an Action section to model action variables, and a Rule Bind section to link the rule in the rule table.

Modeling Condition Variables

To model condition variables, we'll make use of attributes of classes that are defined in the Domain Model. We can drag and drop the required attributes from the Project Browser onto the Condition Variable column. In Figure 4-4 above, we've dragged the `age`, `BadHistoryLevel`, and `ValidLicenceNumber` attributes from the `Customer` class into the "Action Variable" column.

Next, we'll define a range of accepted values for each attribute (such as allowable `Customer.age` values being between greater than 18). A new constraint, `AllowableValues`, is created for the attribute. You can check this constraint by opening the Properties dialog for the attribute and selecting the Constraints tab. If the condition variable references an enumeration, the enum literals are not editable in the Edit Allowable Values dialog.

Modeling Action Variables

In the Action Variable section, when a specific value of a condition variable calls an operation or decision attribute you assign the operation or attribute as an action.

To model action variables, drag and drop the required attribute or operation from a Domain Model Class in the Project Browser onto the Action Variable field. For an attribute, right click on the Allowable Values field and type the range of values in the text box (e.g. Accept, Reject for `Application.Status` in Figure 4-4). Select the appropriate response in the Value column fields. If the dropped action variable is of type enum, the Allowable Values fields are automatically set with the enum literals. For an operation, a checkbox displays in each of the Value column fields.

Binding Business Rules to Conditions

The **Rule Bind** section lies on top of the Condition section. It binds the condition variable and action variable values to the appropriate rule in the Rule Table.

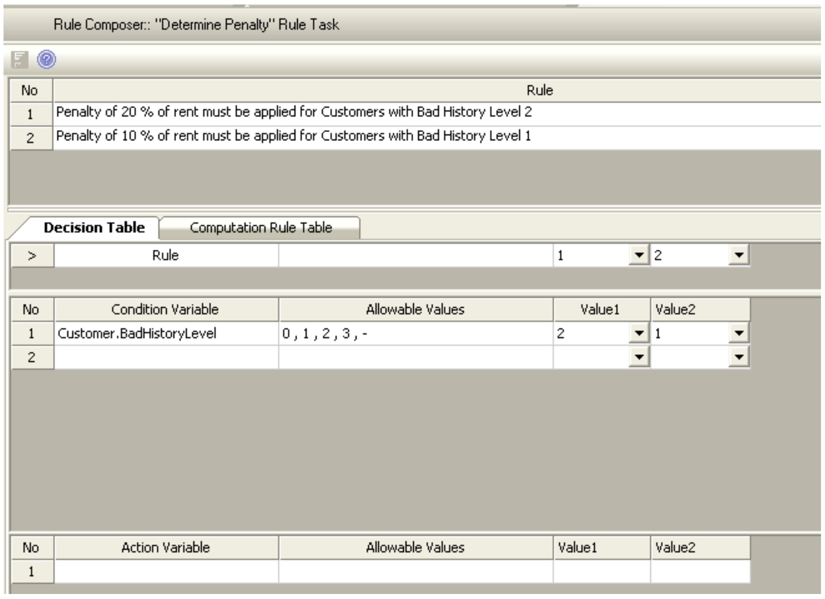
To bind a rule, follow the steps below.

- Select the rule number in the Rule field over one of the Value columns
- Ensure that the values set in the Value<n> field for the condition variables and action variables, underneath the rule number, all satisfy the rule
- Click the Save button

In Figure 4-4, Rule 1 specifies that a ***Car must not be rented to Customers of age less than 18***:

- Select 1 in the Rule field over the Value1 column
- Select < 18 against `Customer.age` in the Value1 column in the Condition table
- Select No against `Customer.Eligible` in the Value1 column in the Action table
- Select Reject against `Application.Status` in the Value1 column in the Action table
- Select the checkbox against `Rent.PostError` in the Value1 column in the Action table.

Figure 4-5 shows the rules and logic for determining penalties based on the `Customer`'s history.



Rule Composer:: "Determine Penalty" Rule Task

No	Rule
1	Penalty of 20 % of rent must be applied for Customers with Bad History Level 2
2	Penalty of 10 % of rent must be applied for Customers with Bad History Level 1

>	Rule		1	2

No	Condition Variable	Allowable Values	Value1	Value2
1	Customer.BadHistoryLevel	0 , 1 , 2 , 3 , -	2	1
2				

No	Action Variable	Allowable Values	Value1	Value2
1				

Figure 4-5. Rule Composer being used to specify the "Determine Penalty" RuleTask.

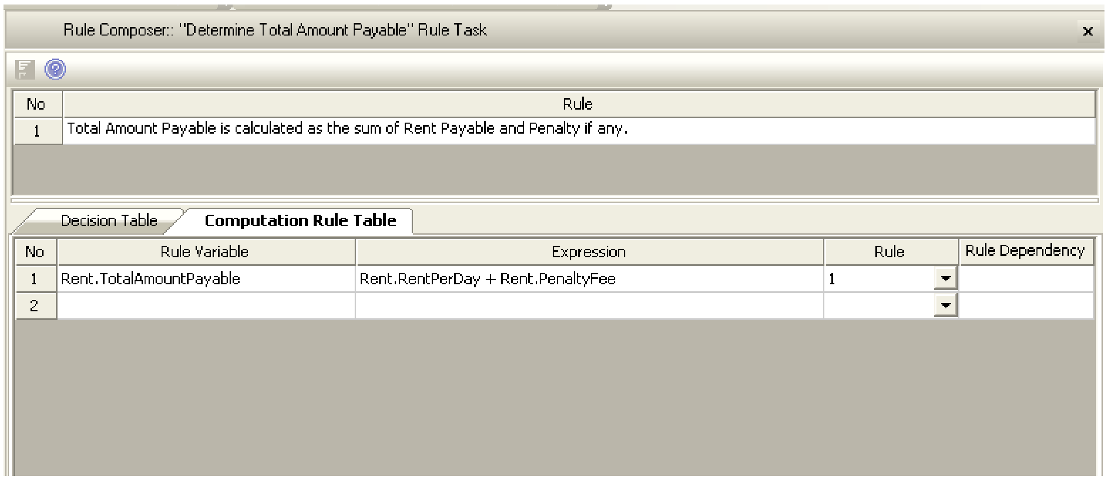
Modeling Computational Rules

The Computational Rule table enables you to model rules involving computations. The table has three following columns: Rule Variable Expression, Rule, and Rule Dependency.

To define a computational rule, follow these steps:

1. Drag and drop the appropriate attribute from a Class in the Domain Model into the Variable field
2. Type the expression to be evaluated
3. Type the rule number from the Rule table of the rule being modeled, to link the table data to the rule

Figure 4-6 shows an example of using the Computational Rule Table.



Rule Composer:: "Determine Total Amount Payable" Rule Task

No	Rule
1	Total Amount Payable is calculated as the sum of Rent Payable and Penalty if any.

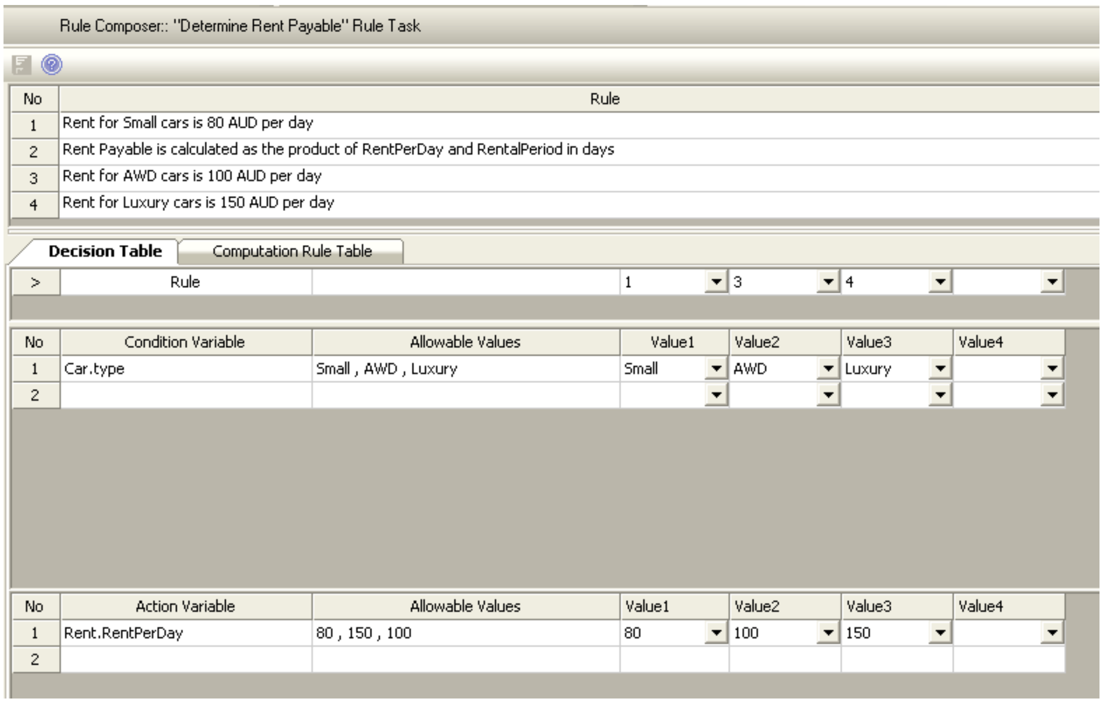
No	Rule Variable	Expression	Rule	Rule Dependency
1	Rent.TotalAmountPayable	Rent.RentPerDay + Rent.PenaltyFee	1	
2				

Figure 4-6. Computational Rule Table for determining Total Amount Payable.

Combining Decision Tables and Computational Rules

It's possible to use the Decision Table and Computational Rule Table together, as shown in Figures 4-7 and 4-8.

If the rule depends on another rule being satisfied first, type the number of that rule in the Rule Dependency field. If the computation rule is also a conditional rule, add the condition variable in the Decision table and bind the appropriate rule in the Rule Bind section.



Rule Composer:: "Determine Rent Payable" Rule Task

No	Rule
1	Rent for Small cars is 80 AUD per day
2	Rent Payable is calculated as the product of RentPerDay and RentalPeriod in days
3	Rent for AWD cars is 100 AUD per day
4	Rent for Luxury cars is 150 AUD per day

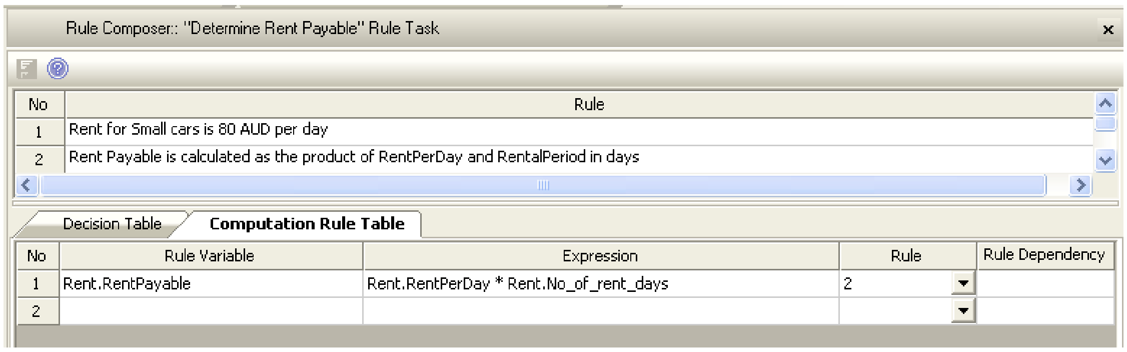
Decision Table | Computation Rule Table

>	Rule		1	3	4	

No	Condition Variable	Allowable Values	Value1	Value2	Value3	Value4
1	Car.type	Small , AWD , Luxury	Small	AWD	Luxury	
2						

No	Action Variable	Allowable Values	Value1	Value2	Value3	Value4
1	Rent.RentPerDay	80 , 150 , 100	80	100	150	
2						

Figure 4-7. Decision Table for Determining Rent Payable.



Rule Composer:: "Determine Rent Payable" Rule Task

No	Rule
1	Rent for Small cars is 80 AUD per day
2	Rent Payable is calculated as the product of RentPerDay and RentalPeriod in days

Decision Table | Computation Rule Table

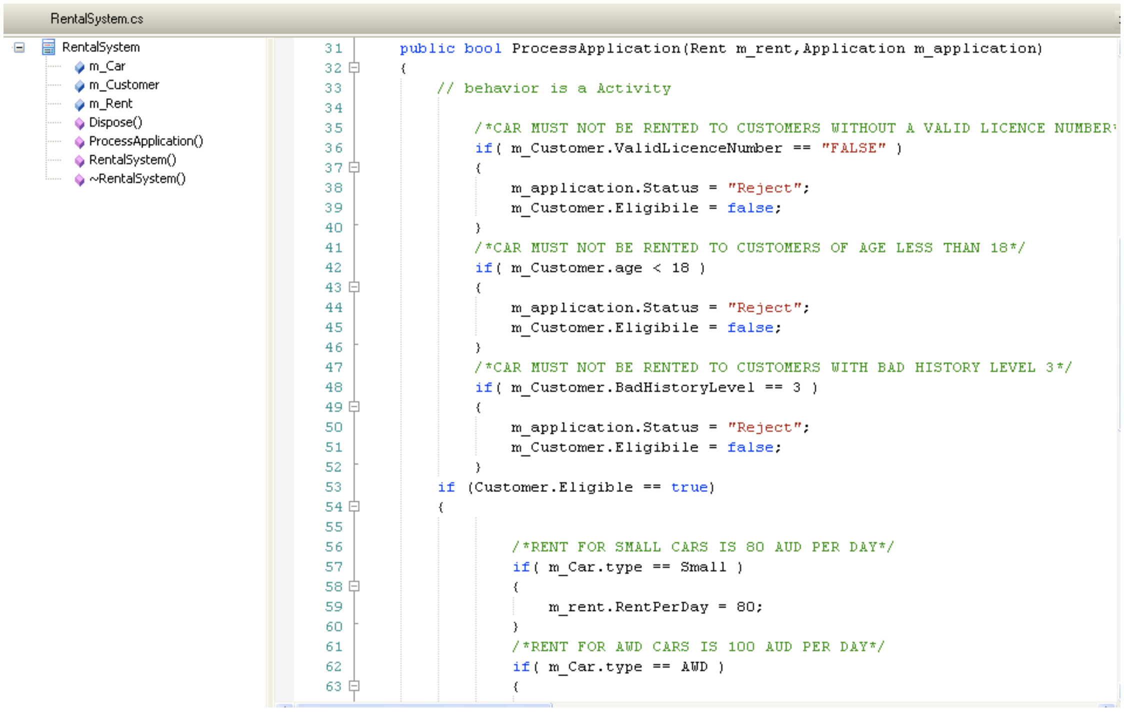
No	Rule Variable	Expression	Rule	Rule Dependency
1	Rent.RentPayable	Rent.RentPerDay * Rent.No_of_rent_days	2	
2				

Figure 4-8. Computational Rule Table for Determining Rent Payable.

Once we've completed specifying the logic for all RuleTasks on our RuleFlow Activity Diagram, we're ready to generate code.

Roadmap: Generate Behavioral Code from Activity Diagrams

Code generation turns out to be very simple (and yields astonishing results) once all the preliminary work has been done. Simply right-click on the Domain Model Class in the Project Browser and select "Generate Code"… and… Voila! No programming required!



```
public bool ProcessApplication(Rent m_rent,Application m_application)
{
    // behavior is a Activity

        /*CAR MUST NOT BE RENTED TO CUSTOMERS WITHOUT A VALID LICENCE NUMBER
        if( m_Customer.ValidLicenceNumber == "FALSE" )
        {
            m_application.Status = "Reject";
            m_Customer.Eligibile = false;
        }
        /*CAR MUST NOT BE RENTED TO CUSTOMERS OF AGE LESS THAN 18*/
        if( m_Customer.age < 18 )
        {
            m_application.Status = "Reject";
            m_Customer.Eligibile = false;
        }
        /*CAR MUST NOT BE RENTED TO CUSTOMERS WITH BAD HISTORY LEVEL 3*/
        if( m_Customer.BadHistoryLevel == 3 )
        {
            m_application.Status = "Reject";
            m_Customer.Eligibile = false;
        }
    if (Customer.Eligible == true)
    {

            /*RENT FOR SMALL CARS IS 80 AUD PER DAY*/
            if( m_Car.type == Small )
            {
                m_rent.RentPerDay = 80;
            }
            /*RENT FOR AWD CARS IS 100 AUD PER DAY*/
            if( m_Car.type == AWD )
            {
```

Figure 4-9. 100% complete logic, with Business Rules appearing as comments within the code!

It's worth examining the code shown in Figure 4-9 quite carefully. Here are a few points worth noting:

- **The entire RuleFlow diagram is code generated as if it were a single class Operation on the DomainModel Class**
- **Each RuleTask is expanded in turn**
- **Within each RuleTask, the Business Rules are automatically propagated forward into the code as comments**
- **Attribute and Operation names are taken directly from the Rule Composer**
- **No manual programming intervention is required**

It doesn't take a whole lot of imagination to see that this capability can be a real "game-changer". Many organizations have thousands of business rules to implement, and "errors in translation" between Subject Matter Experts, Business Analysts, and Programmers are the norm, not the exception. Many of those errors can be eliminated using Behavioral Code Generation and the Business Rule Composer.

There's one more step in our Roadmap for "business-rule centric" processes, and that's the process of building the system in either Eclipse or Visual Studio. That's the subject of the next chapter.

Chapter 5

Integrating Models and Code

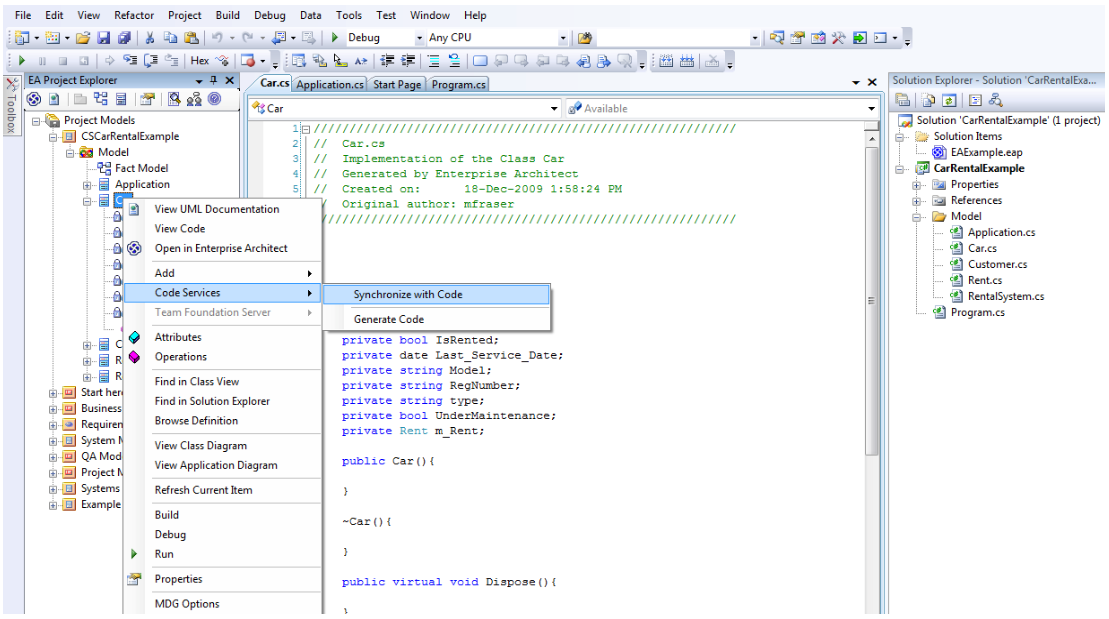
Enterprise Architect contains numerous features to help with code generation and reverse engineering, and also integrates closely with the *Visual Studio* and *Eclipse* development environments via its MDG Integration technology. Many of Enterprise Architect's code engineering capabilities, including forward and reverse engineering, and Enterprise Architect's powerful code template framework, are described in detail in the Enterprise Architect for Power Users multimedia tutorial.[7] This chapter will focus in on the MDG Integration capability.

Mind the Reality Gap

Since the beginning of modeling time, the gap (sometimes a chasm) between models and code has always been problematic. Models, the argument goes, don't represent reality… only the code represents reality… therefore the model must be worthless, and we should just skip modeling and jump straight to code. Those who have used this argument to avoid modeling probably felt quite safe in doing so because nobody has ever managed to make "reverse engineering" or "round-trip engineering" a very seamless process… until now. The innocuously named "MDG Integration" product changes the whole equation.

Bringing UML to the IDE

You can lead some programmers to UML, but you can't always make them embrace modeling. The ever-present gap between models and code is one of the reasons for this. Modeling introduces another environment, another tools interface, another user interface to learn, and forces the programmer to leave the familiar confines of his or her coding environment, where he has all the comforts of home.



```
///////////////////////////////////////////////////////////////
//  Car.cs
//  Implementation of the Class Car
//  Generated by Enterprise Architect
//  Created on:      18-Dec-2009 1:58:24 PM
//  Original author: mfraser
///////////////////////////////////////////////////////////////

    private bool IsRented;
    private date Last_Service_Date;
    private string Model;
    private string RegNumber;
    private string type;
    private bool UnderMaintenance;
    private Rent m_Rent;

    public Car(){

    }

    ~Car(){

    }

    public virtual void Dispose(){

    }
```

Figure 5-1. Bringing the UML model inside the IDE (in this case, Visual Studio) has many benefits.

[7] See: http://iconixsw.com/EA/PowerUsers.html

But what would happen if the UML model was brought inside of the programming environment? (See Figure 5-1). Let's say if you could open your project, right click a menu, and say something like "Attach UML Model". So you can browse your use cases, sequence diagrams, classes, etc from within *Visual Studio* or *Eclipse*.

Then let's suppose you could hot link a package of classes from the UML window to the source code. Nice, but not compelling yet? How's this? You can double-click an operation on a class in the UML window and instantly browse to the source code for that method, and you can edit the code as you normally would in *Visual Studio* or *Eclipse* and update the UML model by right-clicking on the class and choosing *Synchronize*.

Suddenly, the UML model is actually helping you to navigate through your code, you can click to see the use cases and sequence diagrams that are using the classes you're building, and you can re-synch the models effortlessly. Suddenly your UML model is the asset that it was supposed to be all along.

But… here's the six million dollar question: how do you keep the model and the code synchronized over the lifetime of the project?

Four Simple Steps to Modeling Nirvana – Without Chanting OMMMMM

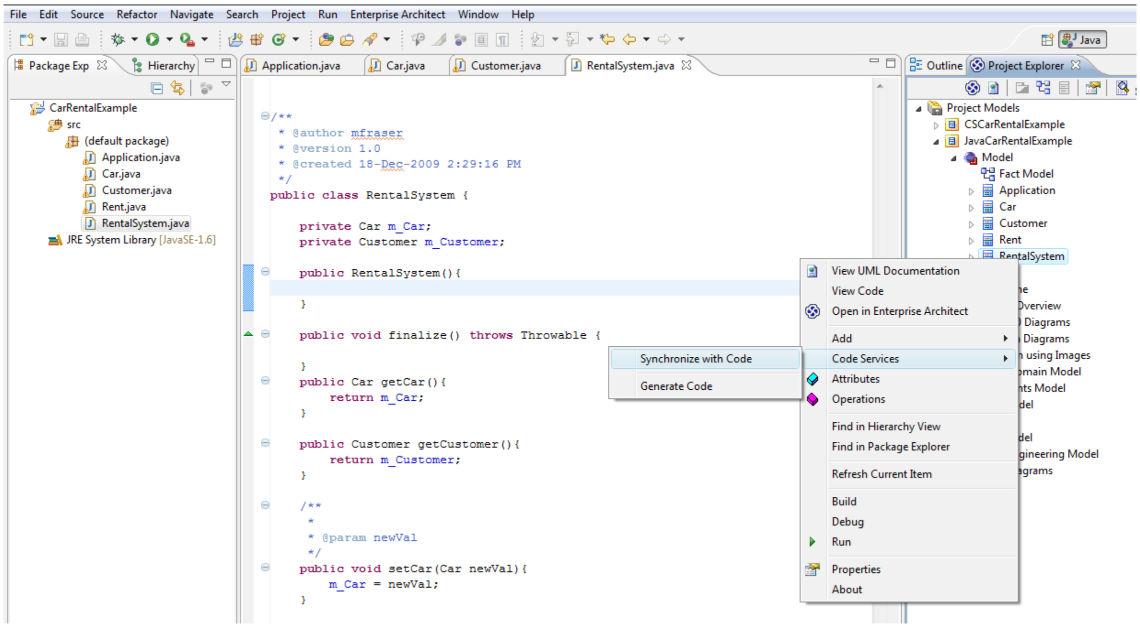
A few years ago, Matt Stephens and I wrote a whole chapter in **Agile Development with ICONIX Process**[8] about how to synchronize models and code, and the reasons why it's important. Synchronizing models and code is still just as important, but the folks at Sparx Systems have obsoleted the "how-to" guidance from that chapter. Now it's absurdly simple. So simple that an old tool-builder like me wonders "why the heck didn't I think of that?"

Here's how it works:

1. Connect your UML model to a Visual Studio or Eclipse project
2. Link a package in the model to classes within the IDE
3. Browse the source code by clicking on operations on classes
4. Edit the source code in your IDE

Enterprise Architect keeps your model and code synchronized, automatically, or you can force synchronization at any time by selecting *Synchronize from Code* from the *Code Services* menu as shown in Figure 5-2.

[8] See http://www.iconixsw.com/Books.html for more on the Agile/ICONIX process.



```java
/**
 * @author mfraser
 * @version 1.0
 * @created 18-Dec-2009 2:29:16 PM
 */
public class RentalSystem {

    private Car m_Car;
    private Customer m_Customer;

    public RentalSystem(){

    }

    public void finalize() throws Throwable {

    }
    public Car getCar(){
        return m_Car;
    }

    public Customer getCustomer(){
        return m_Customer;
    }

    /**
     *
     * @param newVal
     */
    public void setCar(Car newVal){
        m_Car = newVal;
    }
```

Figure 5-2. Java code and UML model for our Car Rental example shown in Eclipse

Wrapping Up

That completes our roadmap for developing Service Oriented Architecture projects using the Enterprise Architect Business and Software Engineering Edition. Our roadmap has taken us from Requirements definition, through implementation of web-service centric scenarios using BPMN, BPEL, and WSDL, business-rule centric scenarios using Behavioral Code Generation and the Business Rule Composer, and finally covered how to effortlessly synchronize UML models and source code over the lifetime of your projects.

ICONIX is available to help with a variety of training and consulting services for your SOA projects. If you think we might be able to help, please contact us at SOATraining@iconixsw.com.

We wish you success in your development efforts!