ASP.NET MVC Web-Tier

Introduction

Codagen® Technology Accelerators™ consist of pre-built transformation templates and UML™ profiles that are ready to use in your development projects. Each Accelerator includes an example that illustrates how you can use Codagen Architect™ to transform a platform-independent model (PIM) into a platform-specific model that represents the source code—a "code PSM."

Purpose

The ASP.NET MVC Web-Tier Technology Accelerator uses Codagen Architect to map a UML platform-independent model (PIM) into the web-tier of a web application following a MVC approach that is targeted toward ASP.NET.

Compatibility

This is Version 1.1 of the Technology Accelerator. It requires Codagen Architect Version 3.0 Service Release 2.

What's New in This Technology Accelerator?

Version 1.1 uses object flows to make the link between the web page classes in the web page class diagram and activities that represent those same pages in the web page navigation activity diagram.

About This User Guide

This intention of this document is to familiarize you with the Technology Accelerator, provide tips on how to apply it to your own projects, and walk you through an example.

Specifically, the guide addresses the following topics:

- Overview of this Technology Accelerator
- Description of input to and output from the Technology Accelerator:
  o Required characteristics of the UML PIM input
  o Extension of the PIM (the architecture specification)
  o Characteristics of the code PSM output
- Transformation of a PIM to a code PSM: description of the code generation templates
- Applying the Technology Accelerator to your own projects
- Transformation example:
  o Files provided
  o Steps to generate executable code
  o Other steps, if necessary, to complete the example

Overview

ASP.NET is a part of Microsoft® .NET® platform and is used to create web applications. It provides the environment and all the necessary support classes to run applications that can render themselves on down-level or up-level browsers, maintain state between sessions (even within a web farm), and more. ASP.NET does not dictate how an application should be built.

Using the UML model (PIM) as input, this Technology Accelerator will create a web application framework that uses ASP.NET. The scope of the Technology Accelerator is the presentation tier of a web application. The business and data tiers are outside the scope and are assumed to be created separately.

To use industry practices and diminish development effort as well as maintain separation of concerns, we have used the Model-View-Controller (MVC) architectural pattern within the web presentation tier.

The MVC web presentation tier will delegate to the business tier through two facades. These facades encapsulate the entire business tier through the view exposed by the business entities present in the PIM. These business entities do not represent the actual business tier but rather the view of the business tier as needed by the presentation tier to accomplish actions invoked by the user and return meaningful information. As long as the two facades cooperate using the data and actions exposed in the business entities, connection to any business tier is possible.

Figure 1 displays a high-level activity diagram of an interaction between a user and a web page built by the Technology Accelerator.

Figure 1. Web Client Interaction with a Web Page
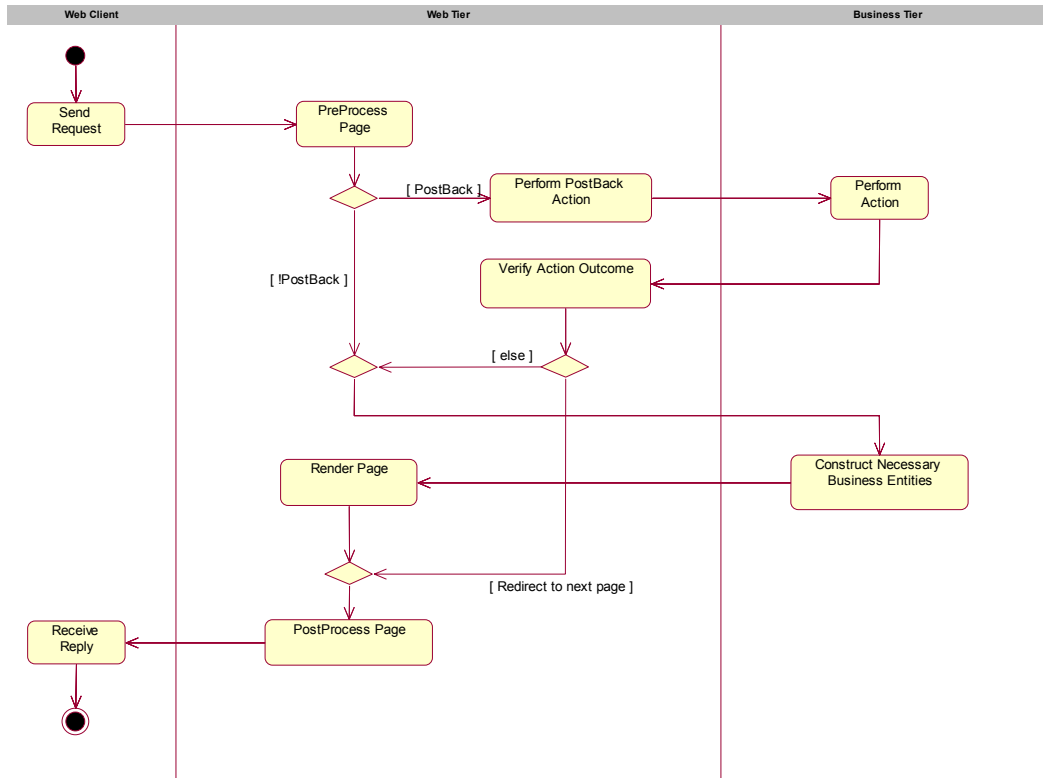
Input and Output: A Black-Box View

This section takes a black-box view—it describes the expected input to the Codagen project and the produced output. Specifically, it describes the nature of the PIM, including its extensions, and the resulting refined physical model that represents the source code: the code PSM.

The focus is on *what* goes in and comes out. A later section describes *how* the transformation occurs.

Platform-Independent Model

Any UML model that possesses the following characteristics is suitable for use with this Technology Accelerator to generate a code PSM.

The model must contain three distinct diagram types:
- Business analysis class diagram
- Web page class diagram
- Web page navigation activity diagram
- Business Class Analysis Diagram

Figure 2 represents an abstraction of the business domain entities required by the web application. As an example, this model would be a first step toward designing a "façade" to permit the web application to use a legacy database system.

Figure 2. Business Analysis Class Diagram
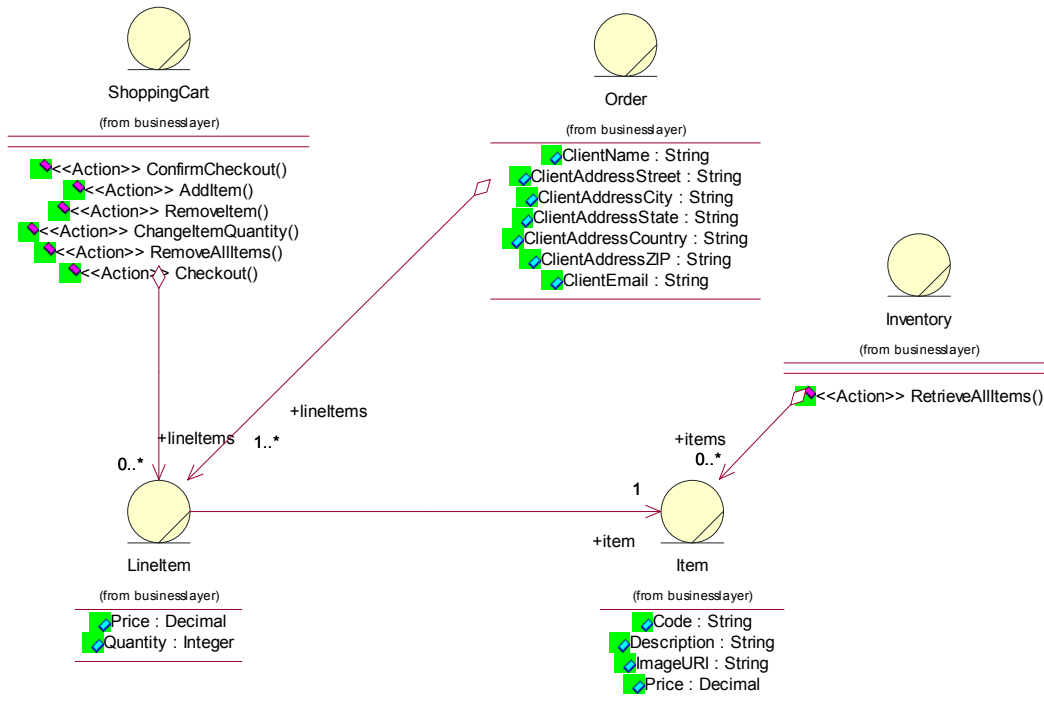
The classes in the diagram represent entities in the business domain (see the stereotype business entity).

Each operation of a business entity represents an action that the web application will perform on the actual business data.

Web Page Class Diagram

Figure 3 represents the web pages in the web application and their links to business entities.
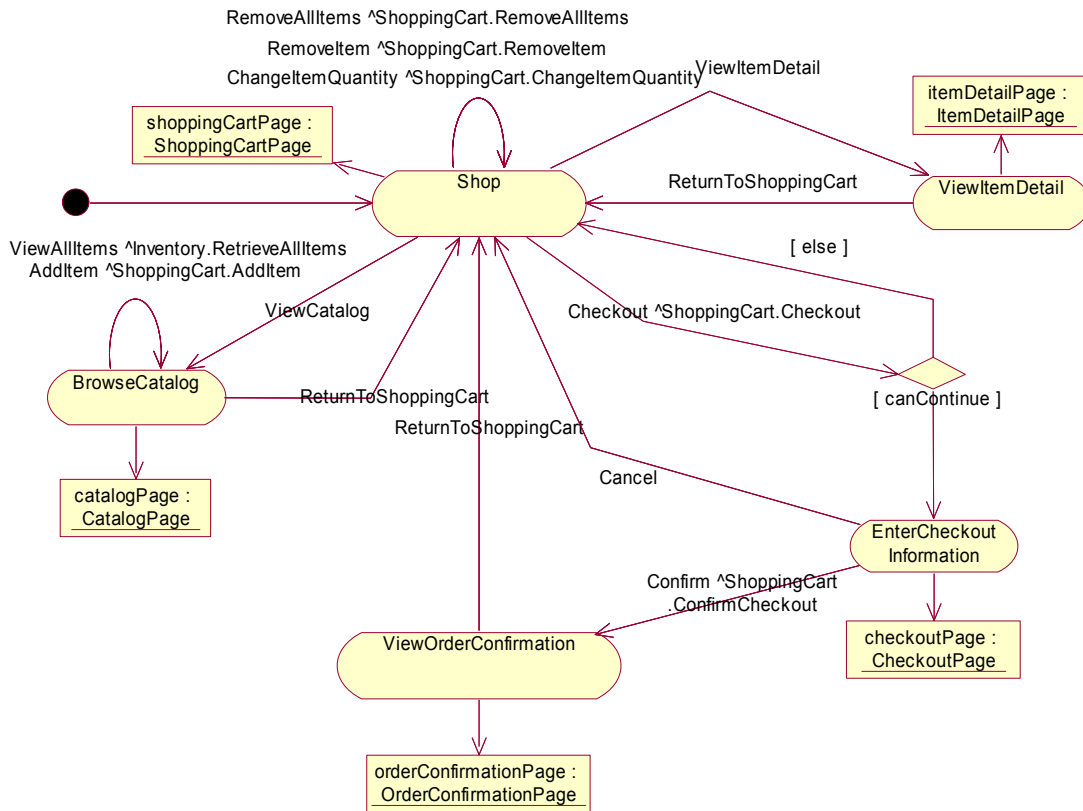
Figure 3. Web Page Class Diagram
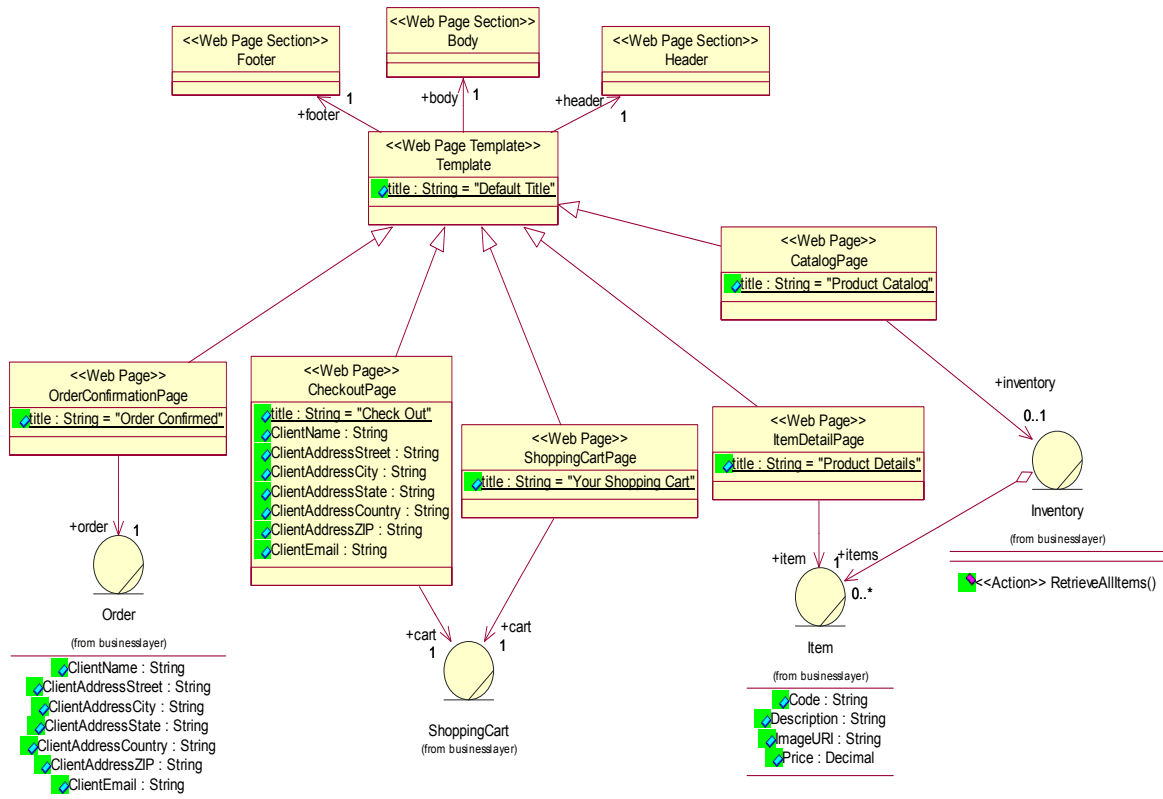
- Each web page in the application is represented as a class (see the stereotype Web Page).
- Web pages that share common layout and other characteristics can be modeled as subclasses of a web page template (see the stereotype Web Page Template).
- Character strings to be inserted into a template are modeled as String attributes of the web template base class. The attributes' initial values represent the values displayed in the page. Web page classes may override these attributes to specify the value they require.
- Areas of a page to be inserted into a template are modeled as page sections linked to the web page template by navigable associations (see the stereotype Web Page Section). The special section identified as the "body" represents the area where the contents derived from the sub-class web pages are displayed.
- Each web page that displays business entity-related information can indicate this dependency in the diagram with an association relationship between the web page and the business entity from the Business Analysis Class Diagram.
- Web page form fields that do not come from the business entities may be modeled by attributes or associations of the web page class.

Web Page Navigation Activity Diagram

Figure 4 represents the possible web page navigations in the web application.

Figure 4. Web Page Navigation Activity Diagram

- Each "Web Page" class in the Web Page Class Diagram must be represented by an activity in the activity diagram. The activity name does not have to be the same as the web page class name.
- To link an activity that represents a web page to the corresponding "Web Page" class, the activity must have an outgoing object flow that terminates in an object that is an instance of the web page class.
- Each transition between web pages (activities) must indicate the request (event) that initiates this transition. If the transition invokes an action on the server, this is represented by a "send event" on the transition. Each "send event" must correspond to an "Action" operation in the Business Analysis Class Diagram.
- Each transition may also contain a guard condition to support conditional transitions between web pages (using the "branch" entity for multiple transitions that differ only in the guard condition). The guard condition uses either the current state of the web application or the result of the "send event," as we assume that all web activities are performed on the server side for now.

Architecture Specification

The following table lists the properties that extend the PIM and their associated model elements. These model extensions are defined in the architecture specification file "CTA WebApplication.csf."

Layer/Issue/Property	UML Element	Description
Presentation		Issues related to presentation layer.
Page Navigation		Properties related to page navigation.
isNavigation	StateMachine	True for any state machine that represents web page navigation. All other state machines are ignored for the purposes of generating navigation code.
Page Composition		Properties related to the make-up of pages.
isPageParameter	Association Attribute	Indicates that the association or attribute is a replacement parameter for a page template. For an attribute, the attribute name is the parameter name and the initial value is the replacement string. For an association, the target role name is the parameter name and the target class is the Web Page Section to be inserted. When used in a Web Page Template instead of a Web Page, the attribute or association defines default values that may be overridden by Web Pages that use the template.
isPageSection	Class	Indicates that the class is a Web Page Section.
isPage	Class	Indicates that the class is a Web Page. Note that a Web Page Section named "body" is automatically generated for each Web Page.
isPageTemplate	Class	Indicates that the class is a Web Page Template. Templates define the layout of Web Pages and contain parameters that can be replaced by specific strings or Web Page Sections for each Web Page.
isPageTemplateTitle	Attribute	Indicates that the attribute represents the page title. Used only in the page template class to indicate which parameter(s) should be inserted into the web page title.
isDefaultTemplate	Class	Indicates that the template is the default template for all web pages.

Layer/Issue/Property	UML Element	Description
isBodySection	Class	Indicates a page section that serves as a placeholder in the template for the web page body. This section will not be generated as a JSP. Instead, each Web Page class will generate a page section that will be inserted into the template at the point occupied by this section in the template. A body section serves only to define two things—the "key" that identifies the body section in the template, and "order" value that positions the body in the sequence of sections in the template.
Page Layout		Properties related to the layout of pages.
webControlType	Association Attribute	Type of web control to be used when the attribute or association is displayed in an editing form. <undefined> means it will not be displayed. FixedText means it will be displayed as ordinary text, not a field. Note that associations are displayed only if the target is navigable.
width	Association Attribute	Field width in characters Applies to TextBox (size attribute).
maxLength	Association Attribute	Maximum # characters accepted by the field. Applies to TextBox.
order	Attribute Association	Specifies the order of fields in the web page. Note that attributes and associations come out in separate sequences, even if you try to interleave them by their order values.
isName	Attribute	Specifies the fields of an object that constitute a meaningful name for the object. For example, for a Person object, it could be firstName and lastName. For a Product object it could be productCode. Used when displaying an association to the object.
includeTarget	Role	How to handle target end of navigable association from Web Page. "forDisplay" target object(s) will be displayed in page. "forEdit" target object will be edited in page. Requires max. target cardinality = 1.
includeTargetDetails	Role	If when including the target end of an association in a Web Page (see includeTarget), you can also display a table of detail records linked to the

Layer/Issue/Property	UML Element	Description
		target. The target must have max. cardinality = 1. Navigable associations from the target will be displayed.
webEventType	Transition	Specifies the type of web page element that triggers the transition: a button in the form, a link at the bottom of the page, or a link on an individual item in a table.
CSS		Properties related to Cascading Style Sheets.
Stylesheet	Model	Stylesheet name. Leading '/' and context root will be provided by generated page.
Data Heading	Model	CSS class name for headings for each type of data object.
Data Column Label	Model	CSS class name for column labels in data tables.
Data Row Label	Model	CSS class name for row labels in data tables.
Data Number Cell	Model	CSS class name for data cells containing numbers in data tables.
Data Text Cell	Model	CSS class name for data cells containing text in data tables.
Form Label	Model	CSS class name for form field labels.
Form Mandatory Flag	Model	CSS class name for flag that indicates a mandatory field in a form.
Form Number Field	Model	CSS class name for form number input fields.
Form Text Field	Model	CSS class name for form text input fields.
Section Heading	Model	CSS class name for JSP Section Heading.
Validation		Properties related to input validation.
isMandatory	Attribute	Indicates that this input field is mandatory.
Business		
Business Entities		
isBusinessEntity	Class Parameter Attribute	Used to identify objects that have sense from the MVC View and Controller layers. They capture information that must be present and generable from the MVC Model.
isAction	Operation	Identifies an Action that can be invoked on the MVC.
isActionOutputParameter	Parameter	Describes data that are returned after an Action is performed.
Technology Accelerator		Issues closely related to the

| Layer/Issue/Property	UML Element	Description
		technology accelerator implementations.
Main		Used to group all main templates that drive the generation process.
Support		Used to group all templates that are not directly executed but are executed from the main templates.
JSP WAF		Technology-dependent values for the JSP WAF Technology Accelerator.
defaultLocale	Model	Default locale for the web application.

As a convenience, the following stereotypes are supported by the architecture specification. Use of these stereotypes is optional, as you can achieve the same effect by setting the corresponding properties.

Stereotype	UML Element	Description
Action	Operation	Identifies operations that correspond to Controller Actions. isAction = True
business entity	Class	Identifies business entity classes. isBusinessEntity = True.
Web Activity	StateMachine	Identifies an Activity Diagram that models web navigation. isNavigation = True.

Web Page	Class	Identifies a class that models a web page.
		        isPage = True.
		        isPageSection = True.

Web Page Section	Class	Identifies a class that models a web page section. isPageSection = True.
Web Page Template	Class	Identifies a class that models a web page template. isPageTemplate = True.

Output: The Code PSM

This section describes the main elements of the MVC web application that are produced by the Technology Accelerator for a given PIM. Figure 5 illustrates these elements.

Figure 5. MVC Application and Relationships
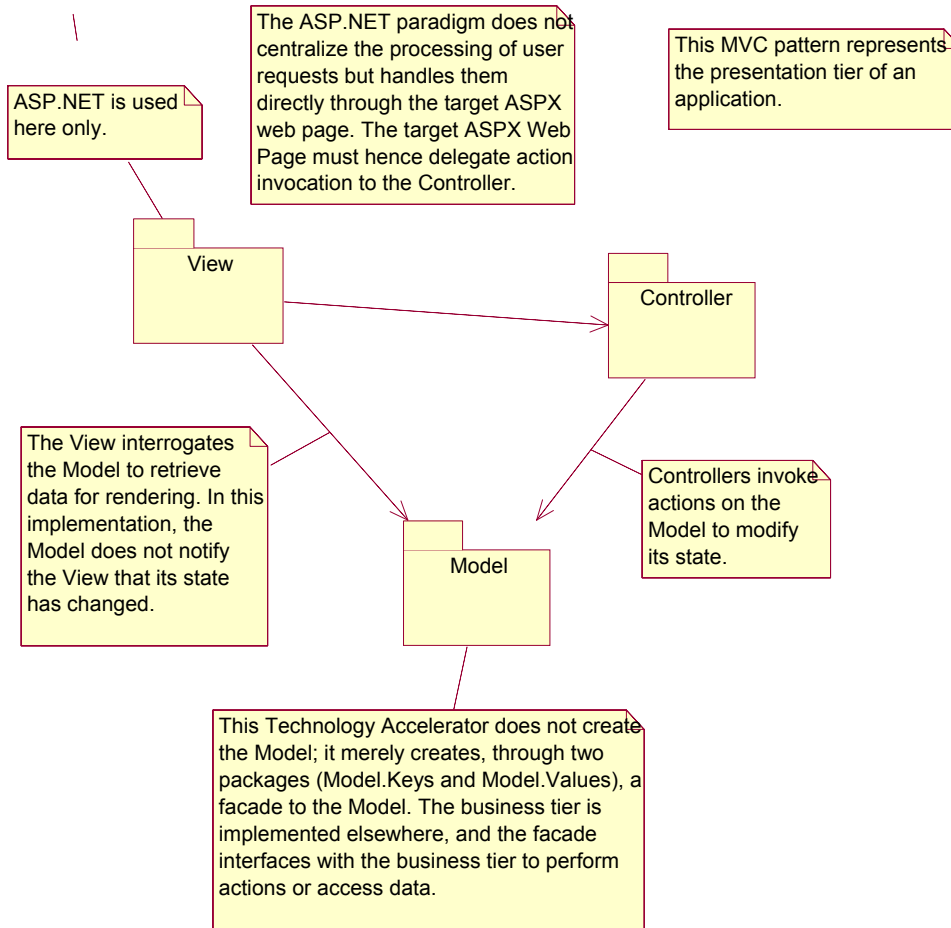
The MVC is separated into three sections:
- Model
- Controller
- View
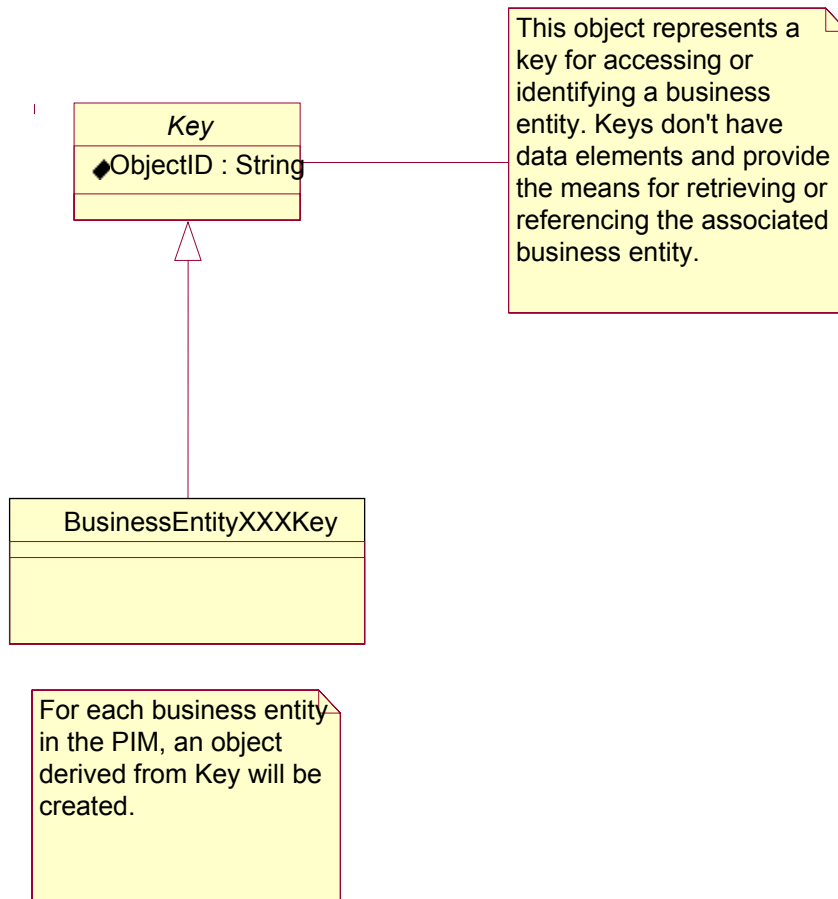
The Model

The Model represents the business tier. As the Technology Accelerator does not generate the business tier, the Model contains two packages, Model.Keys and Model.Values, which represent the view of the business tier that is needed for the application to function correctly. Figure 6 illustrates Model.Keys, and Figure 7 illustrates Model.Values.

Figure 6. The Model.Keys Package
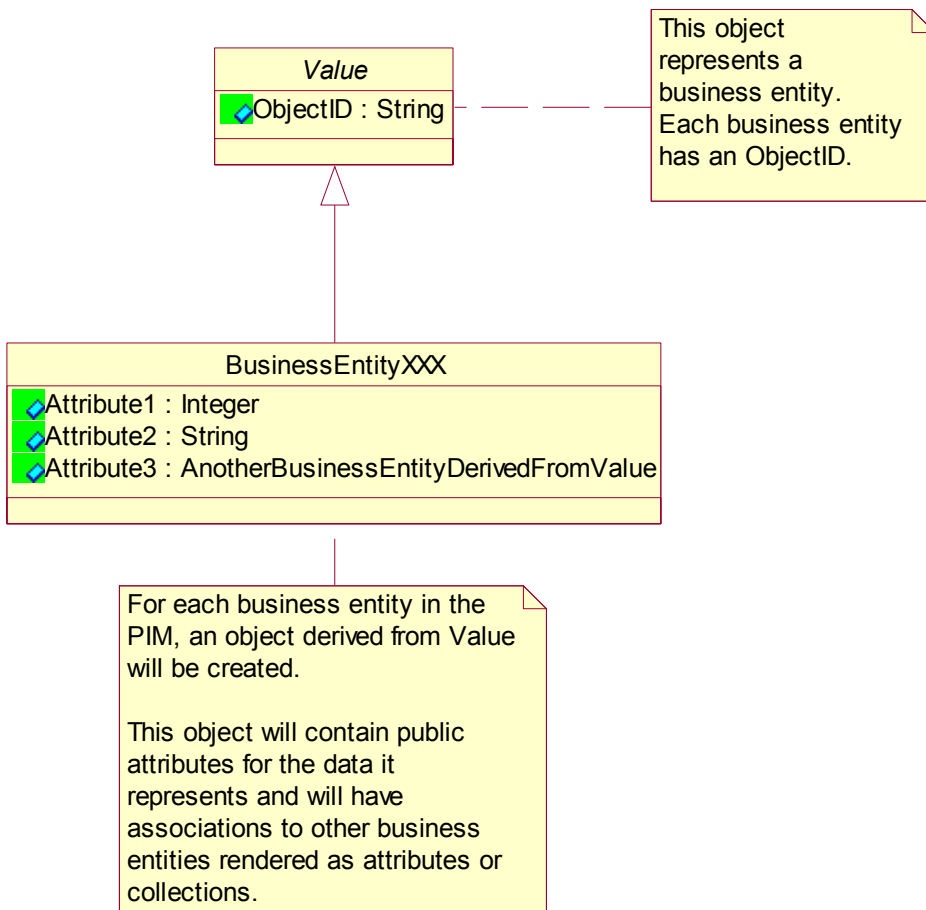
Figure 7. The Model.Values Package
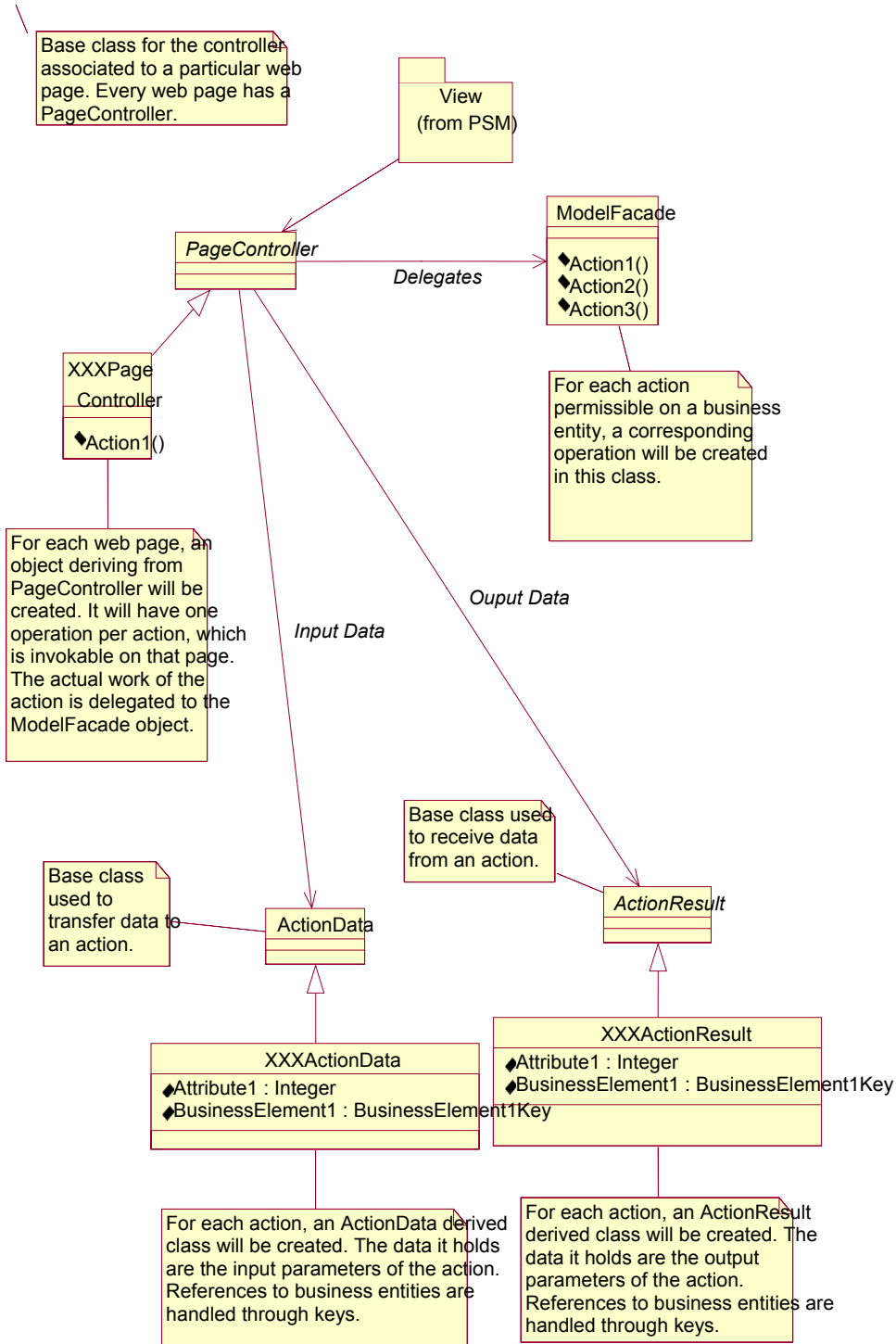

The Controller

The Controller (Figure 8) represents the actions that can be performed on the Model and how to invoke them. The Controller contains an object called ModelFacade, which contains one static operation for each action. Each operation contains a Code Pocket™, which should be filled by a developer, to orchestrate elements on the business tier to achieve the desired result.

Figure 8. The MVC Controller

Base class for the controller associated to a particular web page. Every web page has a PageController.

View
(from PSM)

PageController

Delegates

ModelFacade

◆Action1()
◆Action2()
◆Action3()

For each action permissible on a business entity, a corresponding operation will be created in this class.

XXXPage Controller

◆Action1()

For each web page, an object deriving from PageController will be created. It will have one operation per action, which is invokable on that page. The actual work of the action is delegated to the ModelFacade object.

Ouput Data

Input Data

Base class used to receive data from an action.

Base class used to transfer data to an action.

ActionData

ActionResult

XXXActionResult

◆Attribute1 : Integer
◆BusinessElement1 : BusinessElement1Key

XXXActionData

◆Attribute1 : Integer
◆BusinessElement1 : BusinessElement1Key

For each action, an ActionData derived class will be created. The data it holds are the input parameters of the action. References to business entities are handled through keys.

For each action, an ActionResult derived class will be created. The data it holds are the output parameters of the action. References to business entities are handled through keys.
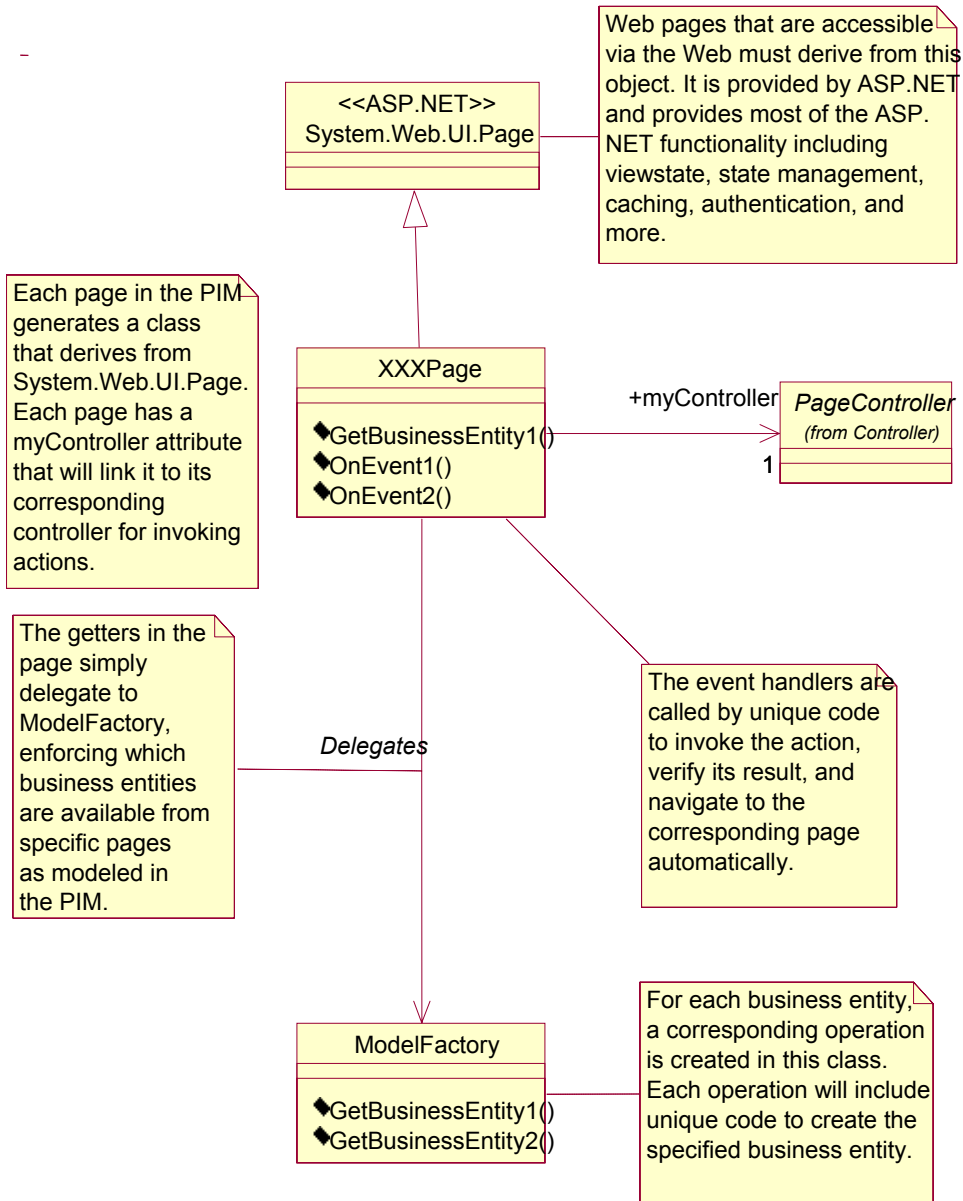
The View

The View (Figure 9) shows the elements that interface with ASP.NET to provide the presentation (GUI) to the user. The user perceives the system as a series of web pages. Some pages can be protected while others cannot. One of the pages is the starting point of the application. Although the client can request any page manually, the page preprocess mechanism must be prepared to handle this case. The MVC View uses HTTP client redirections to perform page changing. This is a bit slower than changing pages on the server but maintains a cleaner application state as viewed by the web browser, as its page cache, backward-forward navigation, and history mechanisms remain coherent.

The View contains an object called ModelFactory, which contains one static operation for each business entity. Each operation contains a Code Pocket™, which should be filled by a developer, to construct a business entity with data that comes from the business tier.

The View contains one object for each web page. Each web page object contains a link to a PageController to invoke individual actions. Each web page object also contains getters for the different business entities that the page has access to (getters delegate to ModelFactory). Finally, each web page object contains event handlers that correspond to the transitions in the navigation activity diagram. These event handlers invoke the action and act upon the result to cause navigation to the proper page.

Figure 9. The MVC View

Web pages that are accessible via the Web must derive from this object. It is provided by ASP.NET and provides most of the ASP.NET functionality including viewstate, state management, caching, authentication, and more.

<<ASP.NET>>
System.Web.UI.Page

Each page in the PIM generates a class that derives from System.Web.UI.Page. Each page has a myController attribute that will link it to its corresponding controller for invoking actions.

XXXPage

◆GetBusinessEntity1()
◆OnEvent1()
◆OnEvent2()

+myController

PageController
(from Controller)

1

The getters in the page simply delegate to ModelFactory, enforcing which business entities are available from specific pages as modeled in the PIM.

Delegates

The event handlers are called by unique code to invoke the action, verify its result, and navigate to the corresponding page automatically.

ModelFactory

◆GetBusinessEntity1()
◆GetBusinessEntity2()

For each business entity, a corresponding operation is created in this class. Each operation will include unique code to create the specified business entity.

From PIM to Code PSM: Inside the Black Box

This section describes the principal code generation templates in the Technology Accelerator—the mechanisms that transform the PIM into the web-tier of a MVC-based ASP.NET web application PSM. The focus is on *how* the PIM is transformed into source code.

Template Name	Type	Description
Main	Composite	Simply serves to start the Technology Accelerator.
ASP.NET MVC Codagen Technology Accelerator	Composite	Actually drives the Technology Accelerator. Logs the version and the Technology Accelerator name and then coordinates the creation of all necessary elements.
MVC Model	Composite	Creates everything that is related to the Model part of the MVC.
MVC View	Composite	Creates everything that is related to the View part of the MVC.
MVC Controller	Composite	Creates everything that is related to the Controller part of the MVC.
ASP.NET Specific	Composite	Creates everything that is specific to ASP.NET.
MVC View \ Create Business Entities Getters	Composite	For each Business Entity that can be accessed by a web page, creates the corresponding getter as well as the associated ASPX testing code.
MVC Controller \ Fill ModelFacade and Create ActionData and ActionResults	Composite	For each action, creates the input and output structures as well as the corresponding operation in the ModelFacade class, which must be filled by unique code to modify the state of the business tier.
MVC Model \ Create Model Keys and Values	Composite	Creates the facade for the MVC Model, which corresponds to an abstraction of the business tier.
MVC View \ Fill ModelFactory	Composite	For each Business Entity, creates a corresponding operation that must be filled by unique code to create the corresponding Value derived object according to a Key.
MVC View \ Create CodeBehinds	Composite	Creates, for every web page, the CodeBehind file and its PageController attribute. Also create its navigation links.
ASP.NET Specific \ Create ASPXs	Composite	For each web page, creates the ASPX file.
MVC View \ Create Navigation Events	Composite	Creates, for each transition a web page can have, an event handler that will perform action invocation, results checking, and page navigation. Also creates the test ASPX file that associates the actions testing block to a web page.
MVC Controller \ Create Page Controllers	Composite	Creates a PageController descendent for each web page, and inserts within it the relevant action invocation operations.

Applying the Technology Accelerator to Your Own Projects

Typically, you will use the supplied projects as the starting point for your own ASP.NET MVC Web-Tier generation projects.

The best way to do this is to create a new C# or Visual Basic .NET project, and import the Technology Accelerator into your project.

Required Files

You will need the following files:

File	Description
Either CTA ASP.NET MVC.gpcs	Codagen Architect project for C#
Or CTA ASP.NET MVC.gpvb	Codagen Architect project for VB .NET
WebApplication.csf	Codagen Architect architecture specification
ASP.NET MVC Web Application.pdf	This document

Configure the .NET Code Integrators

Note that to integrate code from Codagen Architect using the .NET Code Integrators to an "empty web project," you must take special steps to configure Microsoft IIS and Microsoft VisualStudio.NET correctly on your development station.

The following steps assume that your files will be generated in a directory named c:\rd\shoppingcart and that the URL to access it will be http://localhost/shoppingcart.

1. Verify that Microsoft IIS is correctly installed and running on your station.
2. Verify that Microsoft ASP.NET is also installed and configured. It should have had been installed automatically by Microsoft VisualStudio.NET if Microsoft IIS was already installed.
3. Create the c:\rd\shoppingcart directory, into which Codagen Architect will generate all its files.
4. Share the c:\rd\shoppingcart directory as an IIS Virtual Directory named shoppingcart with the following permissions: read, write**,** and directory listing. Note that the directory name and its associated virtual directory name must be the same.
5. Make sure the directories NTFS permissions (if applicable) are set correctly. Invalid configuration could lead to a debugging problem or file-viewing problem that you will have to resolve manually.
6. Set Codagen Architect project properties to the following:

Output folder	c:\rd\shoppingcart
Solution file name	shoppingcart.sln
Project file name	http://localhost/shoppingcart/shoppingcart

7. After the initial code generation, if VB .NET was targeted, before you include the ASPX files in the VS.NET project, blank out the root namespace that is found in the VS.NET project properties.

Create and Develop the Project

The best way to proceed is to create a new C# or VB .NET project, and import the Technology Accelerator into your project.

1. Create your own architecture specification by copying or importing the file WebApplication.csf. You must do this to avoid losing the references to the tagged values when you import the templates
2. Create a Codagen Architect C# or VB project, and import the main template. Importing this template will automatically import all the other templates it uses
3. Adapt the imported templates—renaming them if desired, deleting any that you no longer need, and revising their logic to suit your needs. Change the conditions to select the classes from which you want to generate source code.
4. Add your own templates to generate additional code for your application and environment

A Working Example

To illustrate how to use this Technology Accelerator in your development projects, we have provided an example of an application-specific PSM (code PSM) for you to generate including the necessary files and step-by-step instructions. This Technology Accelerator uses a simple shopping cart example.

The *ShoppingCart* business entity gathers the user's current purchases. The user can add *Item*s from the Inventory to the *ShoppingCart*. When a *ShoppingCart* is purchased, an *Order* is created out of it, and the *ShoppingCart* is deleted.

Web pages are created to display the contents of one or more of these business entities. In addition, navigation between the web pages is made possible by events that might trigger actions that will act upon these business entities.

Files Included in the Example

So that you can generate and compile executable code, the following files are included:

Rose\ShoppingCart.mdl	The Rational Rose model for the shopping cart example
Example\CS	If your target language is C#, the unique files that you must copy to the generation directory to make the shopping cart example fully functional
Example\VBNET	If the target language is VB.NET, the unique files that you must copy to the generation directory to make the shopping cart example fully functional

Generate the Example

This section describes the additional steps required to generate the PSM model, compile the resulting code, and execute the sample application. To assist you in these steps, the example includes a Rose model of the shopping cart. This will permit the generation of all structural code and a sample GUI that allows you to test the sample application.

Also included is a simple business tier linked to a data tier. The data tier is not persisted to persistent storage; all of the data live in the ASP.NET process. The inventory, upon initial usage, randomly creates 100 items, whose codes are named "itemXXX," where XXX is the item number. The business tier validates data and handles errors but is not

transacted, that is, its state is not rolled-back if an unhandled error is fired that might leave the data tier in an unknown state.

To generate the code PSM, follow these steps:
1. Open the Rose model (Rose\ShoppingCart.mdl)
2. Right-click on the shoppingcart package, and launch **Codagen Architect>Implement**.
3. Open either the CTA ASP.NET MVC.gpcs or the CTA ASP.NET MVC.gpvb project.
4. Configure the project as described in the section "Generate the Example" (output directory and so forth).
5. To generate the code PSM, click **Generate All** .

Enhance the Generated Code

You can test the shopping cart without adding any code at all (see the next sections), but the system will throw exceptions where critical Code Pockets™ should be populated. For the shopping cart to become functional, simply add the unique code files to the generation directory (replace existing files) as mentioned in the following sections.

Compile the Code

Code generated by the Technology Accelerator should not encounter problems when compiling. If it does, this might be due to the usage of types that are not referenced automatically by the project.

1. For correct compilation to occur, add references to the following assemblies: System and System.Web.
2. Then link the ASPX files to the project.
   When added to the project, they will automatically become associated to their code behind files.
3. To use the unique code files, copy them to the generation directory, replacing any existing files. Then add the new files to the project.

Test the Code

The generated sample user interface is easy to use and simply demonstrates how to use the generated code. It is intended for testing purposes only. The example will create enough of the GUI for you to test two things:

1. To test the actions of a specific page, navigate to that page. Provide the correct input data, and press **invoke**.
   The result of the action will be displayed.
2. To test the code to retrieve a business entity, navigate to a specific page. Once the entity is retrieved, its **ToString()** operation is called to display a textual description. By default, this operation displays the complete name of the object. This behavior can be overloaded depending on your business entities.

In both cases, typing **null** (in a C# project) or **Nothing** (in a VB.NET project) will initialize the corresponding variable to no references.

If you wish to create a user-friendly custom GUI, simply copy the unique code included with the Technology Accelerator This will allow you to use the shopping cart as if it were a real, production-quality web site.

References

Pattern-Oriented Software Architecture–A System of Patterns, pp.125-143, Bushmann et al., Wiley 1996

Copyright and Trademark Information