Foreword by Sam Guckenheimer

Domain-Specific Development

with Visual Studio DSL Tools

Microsoft .net Development Series

Steve Cook
Gareth Jones
Stuart Kent
Alan Cameron Wills

Domain-Specific Development with Visual Studio DSL Tools

4
Presentation

Introduction

Chapter 2 introduced the different aspects of the definition of a DSL: the domain model; the presentation layer, including graphical notation, explorer, and properties window; creation, deletion, and update behavior; validation; and serialization. Chapter 3 described the first aspect, the domain model. This chapter describes how to define the presentation aspect, that is, how information encoded in the underlying model elements gets presented through the UI of the designer. There are three windows in the UI where information is presented: the design surface, the model explorer, and the properties window. Definition of the presentation aspect therefore involves the definition of the graphical notation used on the design surface, customization of the appearance of the explorer, and customization of the appearance of the properties window.

By far the most complex part is the definition of the graphical notation, so most of the chapter is set aside for explaining that aspect. The explorer and properties window are dealt with at the end.

The Issue State and Issue Project examples introduced in Chapter 2 and downloadable from www.domainspecificdevelopment.com are used to illustrate the concepts and are supplemented by designers built directly from the standard DSL templates shipped with the product. Four language templates have been shipped: Minimal Language, Class Diagrams, Component Models, and Task Flow. Together they exercise most of the graphical

notation supported by the DSL Tools. We'll be careful to indicate which template we've used, where appropriate, and if you wish to try it out in the tools, all you need to do is create a test language using one of those templates.

> **TIP Creating a test language from a language template**
>
> As explained in Chapter 2, when you create a new DSL authoring solution using the DSL designer project template, you will be asked to choose a language template on which to base your new DSL. You can create a test language using this method if you want to try out the examples used to illustrate the text.

A couple of other samples, including a Circuit Diagrams DSL (for generating code from electronic circuit diagrams), are used to illustrate some of the code customizations. These are downloadable from the same location.

Graphical Notation—Overview

A graphical designer presents some elements of a model on a design surface through a graphical notation that uses shapes and connectors. Figure 4-1 shows an Issue State model where a number of issue states and one start element have been created, and where the "Start Element" property for the ***Raised*** issue state is set to be the start element **StartElement1**. All the elements were created through the default menus on the explorer, and the shapes were created on the diagram automatically, as was the connector when the "Start Element" property was set. In general, the creation of a model element automatically causes creation of the mapped presentation element (shape or connector).

The shapes on the diagram are mapped to the elements in the model, as shown in Figure 4-2. The connector maps to the link between the **StartElement** instance and the **IssueState** instance with the name ***Raised***.

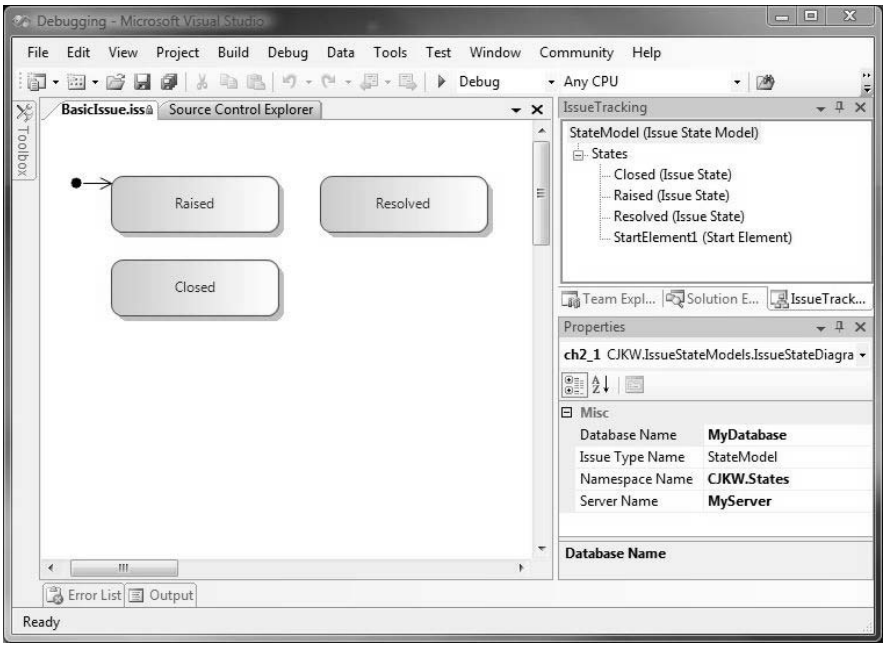

FIGURE 4-1: Issue State designer showing the presentation of a model

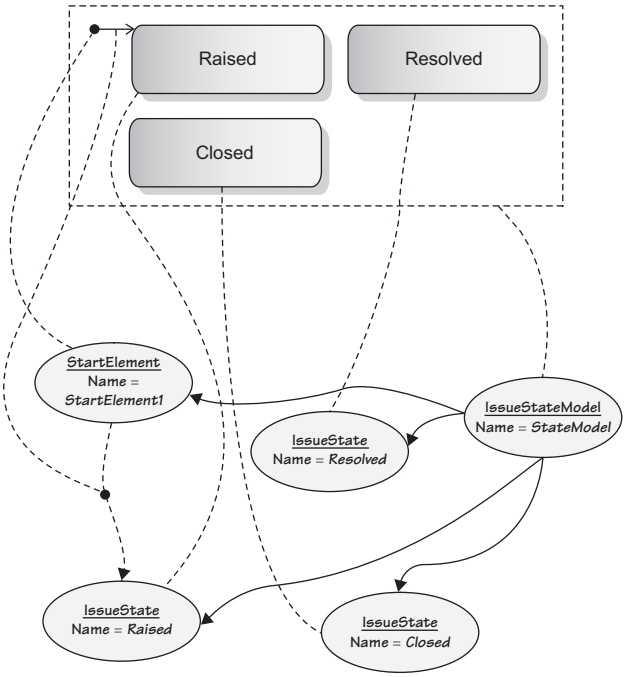

FIGURE 4-2: Maps between diagram elements and model elements

Shapes and connectors in the DSL definition have various properties that allow you to customize color and style, the initial size when the shape is first created, the geometry of the outline of the shape, and so on. You can also define text and icon decorators for shapes and connectors. A text decorator is used to show text, usually the value of some underlying domain property, within or adjacent to a shape; in this example, the **Name** domain property of each state is displayed as a text decorator in the middle of its shape. An icon decorator is used to display a small image whose visibility may be tied to the value of a domain property.

To enable model elements in your DSL to be visualized on the diagram surface, you need to define (a) the kind and appearance of shapes that will be used to present those elements, and (b) a mapping from the shape definition to the domain class of the elements, which dictates both the placement behavior of a shape and how the appearance of decorators is affected by data change. To enable links to be visualized on the diagram surface, you need to define (a) the appearance of the connector that will be used to present the links, and (b) a mapping from the connector definition to the domain relationship for the links, which dictates both the connection behavior of a connector and how the appearance of decorators is affected by data change.

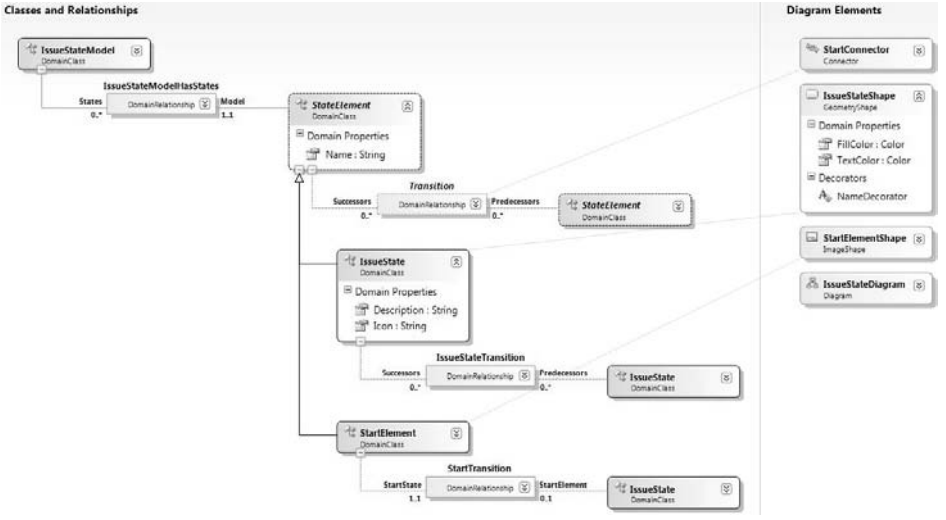

FIGURE 4-3: Definition of diagram elements and maps

Figure 4-3 shows the DSL definition for the Issue State language, created in Chapter 3, with two shape definitions and a connector definition added.

There are several categories of shape; this example uses geometry shapes and image shapes. An image shape displays a specific picture. A geometry shape displays a shape with a geometrical outline such as a rectangle, which the DSL user can resize. In the example, the shape definition called **IssueStateShape** defines a geometry shape that is mapped to the domain class **IssueState**. **StartElementShape** defines an image shape that is mapped to **StartElement**, and **TransitionConnector** defines a connector that is mapped to **Transition**. The effect of this definition on the behavior of the Issue State designer is that whenever an **IssueState** element is created in a model, an **IssueStateShape** representing that element, is created on the diagram. Similarly, for **StartElement**. And whenever a **Transition** link between states is created, a **TransitionConnector** connecting the shapes representing the source and target of the **Transition** link is created. There is also a "Diagram" diagram element (**IssueStateDiagram**), which represents the definition of the design surface itself. This must be mapped to the domain class at the root of the model, in this case, **StateModel**.

Diagram and Editor

The definition of the graphical notation is done in the context of a diagram, which in turn is referenced by the definition of the editor. This section describes these two aspects of a DSL Definition.

> **TIP Creating diagram and editor definitions**
>
> The diagram and editor definitions appear as nodes in the DSL explorer. Both nodes in the DSL definition are created for you when you create a DSL authoring project using the DSL designer project template. If you happen to delete them, they can be recreated by clicking on the top level (Dsl) node in the DSL explorer and selecting "Add Diagram" or "Add Custom Editor"/"Add Designer" from the context menu.

Diagram

Primarily, the diagram definition is a container for the shape and connector maps. Figure 4-4 shows the definition of the diagram for the Issue State DSL.

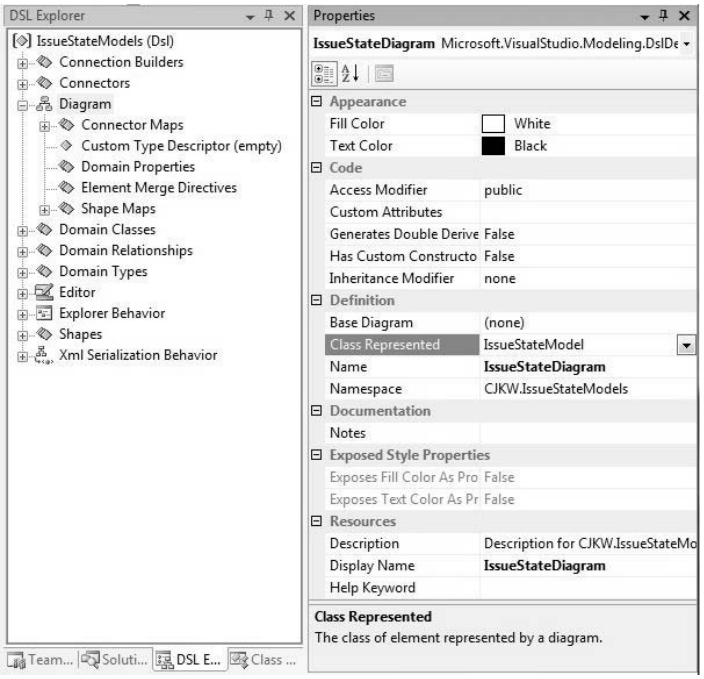

FIGURE 4-4: Definition of diagram for Issue State DSL

The window on the left shows sub-elements of the diagram definition in the DSL explorer, and the window on the right the properties of the diagram definition. The property settings are summarized in Table 4-1.

TABLE 4-1: Property Settings Categories for Diagram

Appearance	Settings that define the visual appearance of the diagram, such as background color.
Code	Settings that influence the form of the generated code for the diagram. All these settings are inherited from domain class, because a diagram definition is also a domain class.