Лекция № 12. Знакомство с DSM-платформой Microsoft DSL Tools.

В этой лекции рассказывается, как задавать проблемно-ориентированные языки (DSLs) с помощью Microsoft DSL Tools, как на основе спецификации языка создавать DSM-пакет.

Microsoft DSL Tools: обо всем помаленьку. Рассмотрим теперь подробно одну из DSM-платформ под названием Microsoft DSL Tools. Этот инструмент является частью пакета Visual Studio 2005 SDK и может быть бесплатно «скачен» с сайта компании Microsoft (http://www.microsoft.com/downloads). Он устанавливается в виде надстройки к Microsoft Visual Studio и позволяет легко создавать DSM-инструменты, специфичные для определенного проекта.

На практике, создавая DSL-пакеты с использованием Microsoft DSL Tools, особенно в начале, постоянно путаешься, где модель, а где – метамодель, что генерируется, а что компилируется, где какие «исходники» и т.д. Чтобы прояснить общую картину и сделать ее более «операбельной», рассмотрим рис. 12.1.

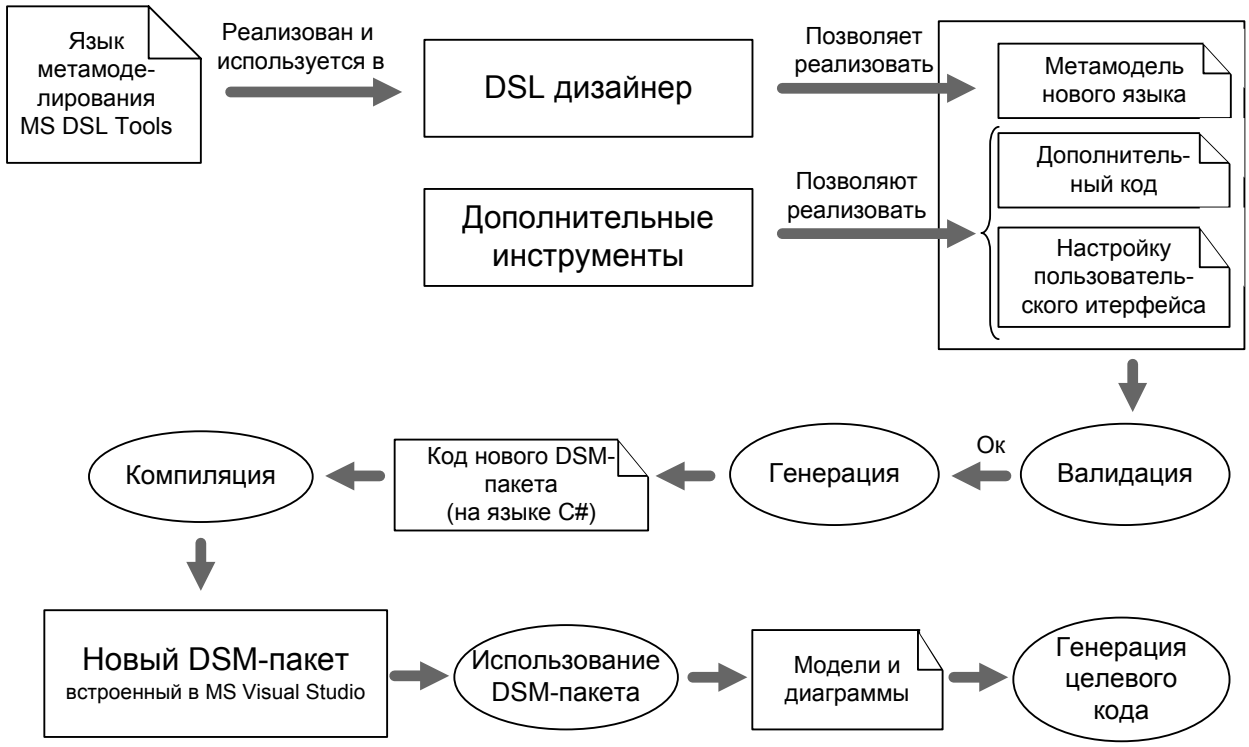

Рис. 12.1

Несколько слов об обозначениях на это рисунке. Овалы обозначают действия, прямоугольники – инструменты, а прямоугольники со скошенным правым углом – данные (подаваемые на вход действию или инструменту и получаемые в итоге). В случае, когда инструменты генерируются, мы обозначаем их не как данные, а как инструменты (прямоугольник «Новый DSL-пакет»). На этой схеме нет ветвлений, поскольку мы изображаем главный сценарий использования DSL Tools.

Теперь обратимся к сути рисунка. Создание DSM-пакета начинается с того, что в Microsoft Visual Studio мы создаем специальный solution под названием Other Project Types\Domain Specific Language Designer. В этом solution автоматически создается два проекта: Dsl и DslPackage. Первый содержит, главным образом, спецификацию метамодели и связанных с ней артефактов, второй – настройки пользовательского интерфейса целевого DSM-пакета. Сгенерированный код DSM-пакета распределяется по обоим проектам.

Главным инструментом Microsoft DSL Tools является DSL дизайнер, Он позволяет создать метамодель нового языка с помощью специального визуального языка (специальный вариант диаграмм классов). Описание этого языка можно назвать мета-метамоделью, поскольку он является средством создание метамоделей новых языков.

Дизайнер «обвешан» большим количеством дополнительных инструментов для задания разных свойств, атрибутов элементам метамодели, а также программного кода на C#, расширяющего стандартные возможности DSL Tools. Часть дополнительных инструментов может вызываться и иным способом, из браузера DSL-solution и используется, например, для задания пользовательского интерфейса целевого DSM-пакета.

Далее, все что мы создали ранее –метамодель, дополнительный код, настройки пользовательского интерфейса – валидируются. Это важно, поскольку потом будет проведена генерация и только после этого – компиляция. Если при компиляции возникнут ошибки, вызванные нашими нестыкующимися друг с другом действиями в SDL дизайнере, то понять, что произошло, в каком месте мы ошиблись, может оказаться не просто. Кроме того, часть ошибок может не «пойматься» при компиляции «перекочевать» в откомилированный код. Тогда целевой наш DSM-пакет будет не правильно работать (непонятно почему!).

Итак, после успешной валидации происходит генерация кода целевого DSM-пакета, а после – компиляция. В результате получается целевой DSM-пакет – графический редактор, броузер, валидатор, генератор кода по моделям и пр.

Этот DSM-пакет появляется в списке инструментов (Items) в Microsoft Visual Studio. Его теперь можно подключить к нашему целевому solution и после этого использовать. С помощью DSM-пакета пользователь может создавать свои собственные модели и диаграммы и генерировать по ним целевой код в рамках своего прикладного проекта.

Разработка метамодели языка. Важнейшим процессом в рамках разработки DSM-пакета на основе Microsoft DSL Tools является создание метамодели нашего языка – абстрактного и конкретного синтаксиса. Тут важно понимать, что во главу угла при разработке DSM-пакета поставлен именно язык – по его спецификации генерируется основной код, прочие спецификации «пристыкованы» с языку.

На рис. 12.2 – 12.4 представлена метамодель нашего языка, который мы приводили в качестве примера в предыдущей лекции.

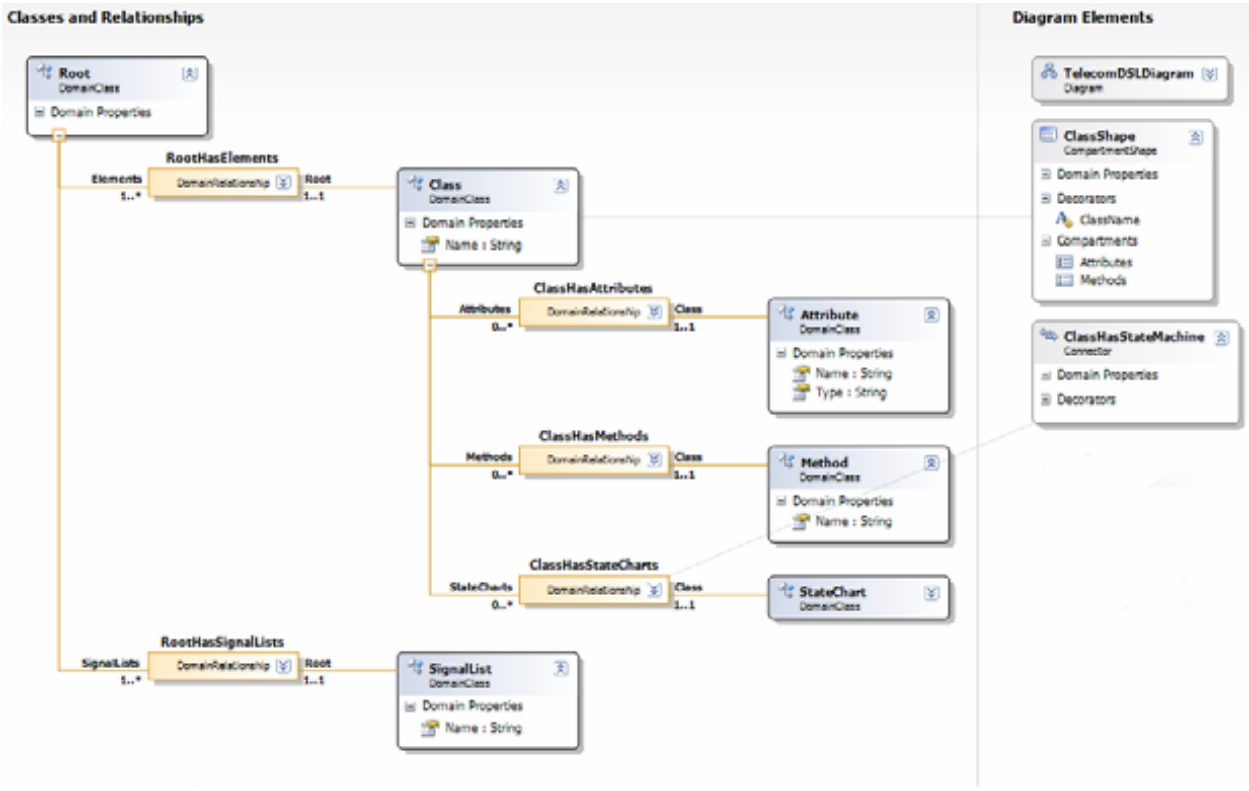

Рис. 12.2.

Рис. 12.3.

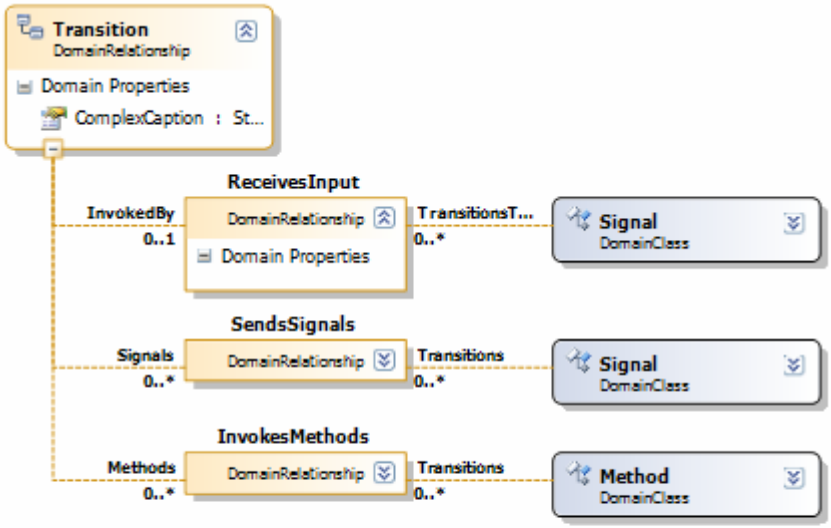

Рис. 12.4.

На рис. 12.5. показывается целевой редактор, сгенерированный по этой метамодели редактор и диаграмма, выполненная на нем (она дублирует рис. 11.6 предыдущей лекции). Рисунки 12.2 – 12.4 представляют одну диаграмму в DSL дизайнере, разрезанную на три части. Эта диаграмма разделена на две горизонтальные области – Classes and Relations и Diagram Elements. Абстрактный синтаксис языка определяется в первой секции, а конкретный – во второй. Класс из первой секции соединяется линией-отношением с классом из второй секции, если определяет конструкцию, которая должна изображаться на диаграммах будущего DSM-пакета.

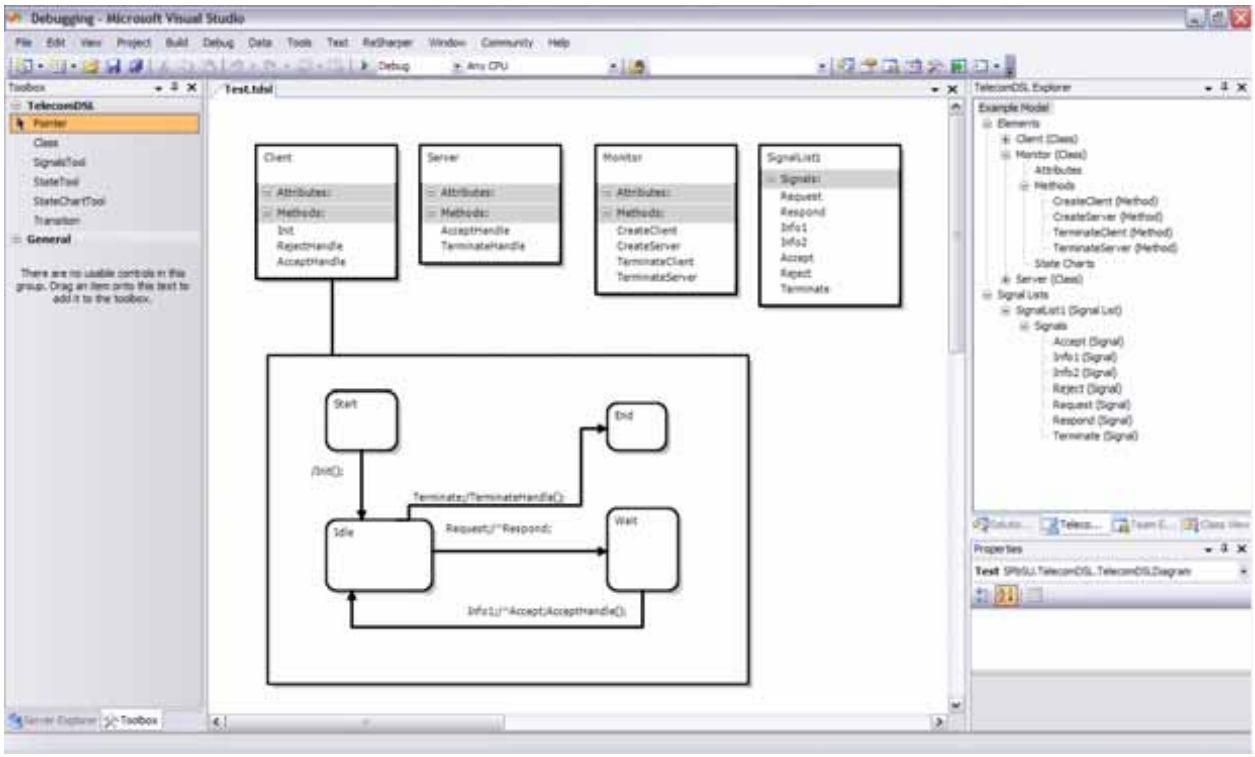

Рис. 12.5.

Мы видим, что эта метамодель существенно отличается от той, которую мы создали для нашего языка на предыдущей лекции. Во-первых, в нашей прежней метамодели не было конкретного синтаксиса. Во-вторых, она была более абстрактной, так сказать, концептуальной, и не содержала многочисленной сопроводительной информации для автоматической генерации DSM-пакета (этой информации почти не видно на рис. 12.2 – 12.4, но она содержится в многочисленных свойствах классов метамодели). В-третьих, есть отличия в нотациях: исходно мы использовали подмножество диаграмм классов UML 2.0, а в DSL дизайнере реализована иная нотация.

Теперь мы рассмотрим основные конструкции графического языка DSL дизайнера.

Доменный класс (domain class) – используется для задания отдельной сущности нашего языка – состояния, списка сигналов и т.д. конструкции элемент абстрактного синтаксиса (конструкция) в создаваемом языке. На рис. 12.2. мы можем видеть доменные классы – Root, Class, SignalList и т.д. Отметим, что метамодель в DSL дизайнере должна начинаться с корневого класса (в нашем случае это класс Root). Все остальные классы располагаются в древообразной иерархии, соединяясь с предыдущим уровнем агрегированием, ассоциацией или наследованием.

Агрегирование (embedding relationship) – используется для задания агрегирования одного доменного класс другим. На рис. 12.2 – 12.4 можно вдеть множество таких отношений, например, RootHasElement. Это отношение имеет множественность 1*:1..1, то есть каждый объект класса Element принадлежит строго одному объекту классу Root, а у одного объекта класса Root может быть много объектов класса Element. Агрегирование (как и другие отношения) изображается на диаграмме специальным боксом, по сути – тоже классом. Это сделано для того, чтобы можно было использовать агрегирование и ассоциации в связях типа многие-ко-многим.

Ассоциация (reference relationship) соответствует обычной ассоциации между двумя классами в UML.

Наследование (inheritance) – соответствует обычному наследованию классов UML.

Теперь поговорим об элементах, с помощью которых в DSL дизайнере задается конкретный синтаксис языка. Они делятся на две группы – фигуры (shapes) и линии (connectors). Предлагается следующие виды фигур:

Геометрическая фигура (geometry shape) задает геометрическую фигуру одного из следующих видов прямоугольник, прямоугольник со скругленными углами, эллипс, окружность.

Сложная фигура (compatment shape) почти во всем аналогчна геометрической фигуре, но позволяет задавать прямоугольники, у которых можно схлопывать/распахивать отдельные секции (например, секцию атрибутов операций у прямоугольника, изображающего класс). Например, как видно из рис. 12.2. – 12.4. графические классы ClassShape и SignalListShape являются сложными фигурами, определяя.

Произвольная картинка (image shape) – способ задать произвольное изображение (в виде, например, картинки в jpg- или bmp- форматах) для конструкции создаваемого в дизайнере языка. Этот тип фигур отличается от сложной и геометрической фигур тем, что

в нем по-другому «отрисовываются» графические атрибуты, например, иконка, различные секции внутри и пр.

Соединитель (connector) используется для задания линий в нотации нашего языка. На рис. 12.4 мы можем видеть соединитель TransitionConnector, который определяет способ изображения линий-переходов между состояниями.

Несколько слов о том, как соединяются конструкции абстрактного и конкретного синтаксиса. Для этого в DSL дизайнере предусмотрена специальная конструкция – *диаграммный соединитель* (diagram element map). На рис. 12.2 – 12.4. мы можем видеть, как доменные классы и отношения (раздел диаграммы Classes and Relationships) соединены линиями с фигурами и соединителями (раздел диаграммы Diagram Elements). Эти линии и есть диаграммные соединители. При соединении доменных классов и отношений с фигурами и линиями у диаграммных соединителей автоматически проставляется соответствие между элементом абстрактного и конкретного синтаксиса. Далее нужно «вручную» проставить соответствие между доменными свойствами и декораторами, а также задать некоторую дополнительную информацию.

Для полноты картины перечислим остальные элементы нотации DSL дизайнера:

- *именованный доменный класс* (named domain class) – доменный класс, который имеет свойство «имя»; как показывает практика, большинство доменных классов имеют имена, поэтому этот класс полезен для удобства пользователей DSL дизайнера;
- *разделитель* (swimlane) – конструкция, которая позволяет задавать различные секции на диаграммах, как например, секции Classes and Relationships и Diagram Elements на диаграммах DSL дизайнера, а также задать, какие графические конструкции в какой секции будут располагаться на диаграммах будущего DSM-пакета;
- *порт* (port Shape) – специальный вид фигуры, позволяющий отображать доменный класс в виде круга на границе фигуры (например, порты на диаграммах компонент в UML 2.0); на рис. 12.6 представлен фрагмент метамодели, определяющий порт, а на рис. 12.7 показана целевая диаграмма, созданная с помощью редактора, где были определены порты в соответствии с рис. 12.6.

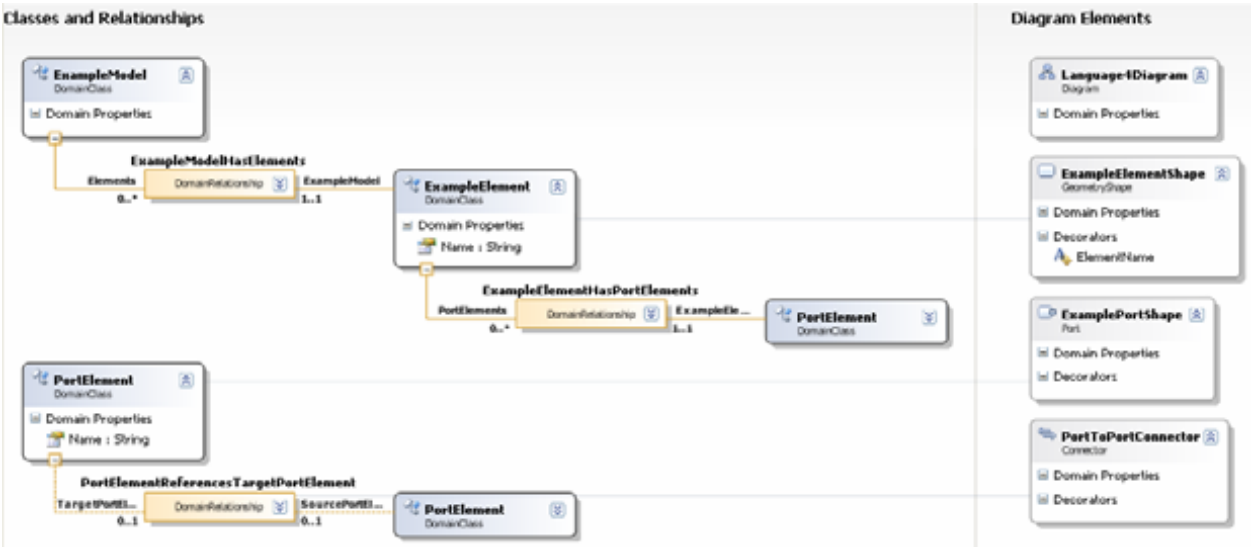

Рис. 12.6.

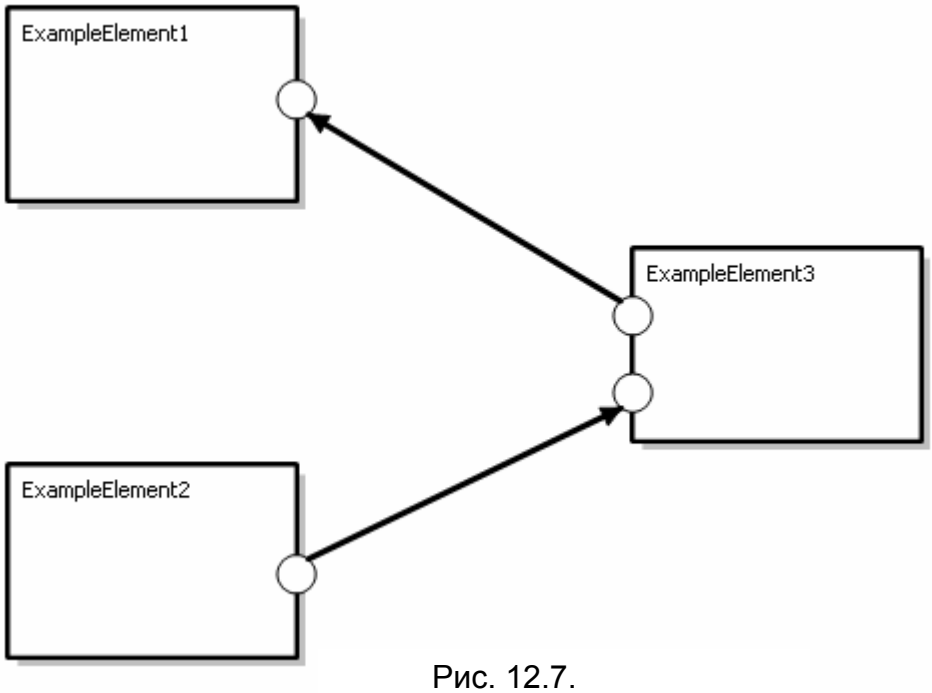

Рис. 12.7.

Все возможные конструкции, доступные в дизайнере, показаны на рис 12.8, слева от рабочей области, в виде списка.

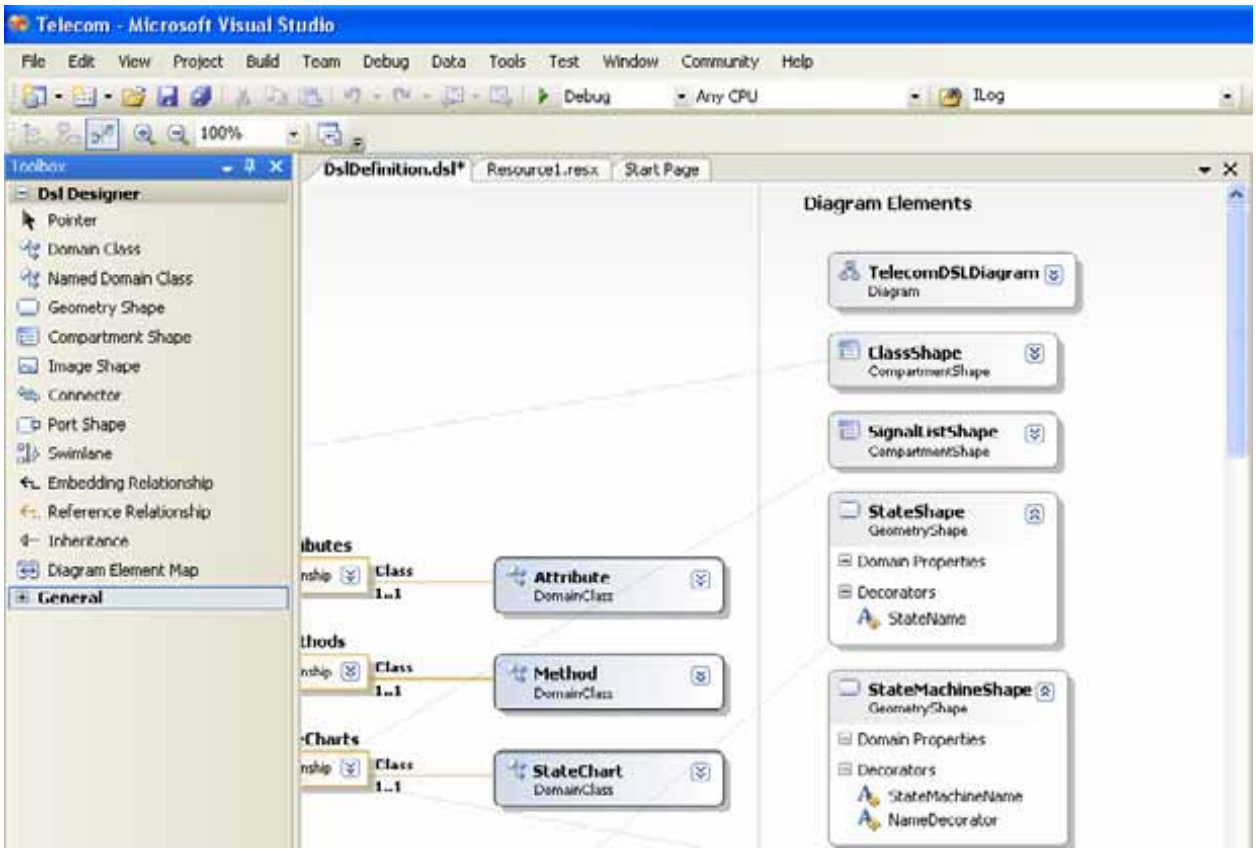

Рис. 12.8.

Обсудим теперь основные свойства элементов языка DSL дизайнера. Для удобства использования они сгруппированы в две основные группы[1]:

- доменные свойства (Domain properties) – есть у всех классов;
- декораторы (Decorators) – есть только у графических классов (фигур и соединителей).

Доменные свойства (Domain Propeties). Создавая в DSL дизайнере класс, разработчик может создать произвольное количество свойств этого вида. Такое свойство имеет следующие группы атрибутов:

- Code – характеристики данного свойства как свойства C#-класса; дело в том, что созданный в DSL дизайнере класс будет превращен при генерации в C#-класса, а его свойства – в C#-свойства этого класса; группа атрибутов code определяет различные C#-характеристики данного свойства, например, его видимость (public protected и т.д.);
- Definition – характеристики данного свойства в метамодели, например, его имя (Name), значение по умолчанию (Default Value), обозначает ли свойство имя класса (Is Element Name) – если да, тогда генератор обеспечит уникальность имен по умолчанию на диаграммах DSM-пакета;
- Kind, например, normal, calculated; в последнем случае у нас имеется вычисляемое свойство, которое задается некоторым кодом на языке C#, прилагаемом к метамодели (то, что на рис. 12.1 обозначено как дополнительный код);
- Documentation – текстовые комментарии к свойству;
- Resources –дополнительные C#-характеристики свойства.

Рассмотрим пример calcutated-свойства. Одно такое свойство присутствует на рис. 12.4. – это свойство метакласса Transition под названием ComplexTransition. Оно задает текст на линии, обозначающей переход между двумя состояниями. Этот текст во-первых, собирается из нескольких доменных классов и отношений, во-вторых, имеет сложное форматирование. Ниже представлен фрагмент кода на языке C#, который определяет правило вычисления и форматирования этого свойства:

```
public partial class Transition
{
  public string  GetComplexCaptionValue()
  {
    string result = string.Empty;
    result += string.Format(@"{0}/{1};{2}{3}",
      this.InvokedBy == null ? string.Empty : this.InvokedBy.Name,
      this.Signal == null ? string.Empty : this.Signal.Name,
                  Environment.NewLine,
                  this.Method == null ? string.Empty : this.Method.Name);
    return result;
```

[1] На самом деле групп несколько больше, да и в рамках этих, рассматриваемых здесь групп, свойства имеют больше атрибутов, чем мы обсуждаем здесь. Напомним, однако, что данная лекция не является переведенным на русский язык справочником по Microsoft DSL Tools. Мы лишь объясняем основные концепции, а для практического использования Microsoft DSL Tools читателю придется еще самостоятельно потрудиться.

```
    }
}
```

Здесь используется механизм частичных классов языка C#, с помощью которого доопределяет класс Transition, автоматически генерируемый по классу Transition. Данный текст помещается в отдельный файл, который помещается в проекте Dsl.

Декораторы (Decorators). Эта группа свойств графического класса содержит определения различных текстовых элементов фигуры или соединителя. Например, у фигуры StateShape на рис. 12.3. есть декоратор StateName, который определяет способ изображения имен состояний, которые мы будем создавать в нашем DSM-пакете.

Декоратор принадлежит одному из следующих типов:
- текст(text);
- изображение (icon);
- схлопывающиеся область (Expand Collapse).

Кроме того, свойства декоратора отличаются для фигуры и соединителя – в одном случае мы имеем фигуру и нам нужно расположить дополнительные текстовые элементы на ней, во втором случае то же самое делается для линии.

Вот основные группы атрибутов декоратора:
- Appearance – настройка шрифтов изображения;
- Definition – имя в коде и в метамодели;
- Layout – задание расположения текста;
- Documentation – примечания метамодели;
- Resources – настройка отображаемого текста.

Настройка палитры. На рис. 12.5, слева, мы видим палитру (toolbox), в которой перечислены все конструкции нашего нового языка, которые мы можем создавать на диаграммах. Для задания палитры мы вызываем соответствующий инструмент DSL Tools. Активируя элемент Edit/Toolbox в браузере для нашего solution мы вызываем инструмент для задания палитры: мы создаем для нее элементы, связываем их с классами метамодели языка, задаем иконку и другие свойства. Аналогичным образом настраивается и браузер целевого редактора.