Appendix I. Metamodel Configuration Offered by Microsoft DSL Tools

Microsoft DSL Tools allow configuring the domain models conceived with the tool.

The file containing the definition of the DSL (the file *DslDefinition.dsl*), as well as the files containing the code generated from text templates (the files *DomainClasses.cs* and *DomainRelationships.cs* generated from the files with the same name but with extension *.tt*) are all in Solution Explorer shown in Figure 46.

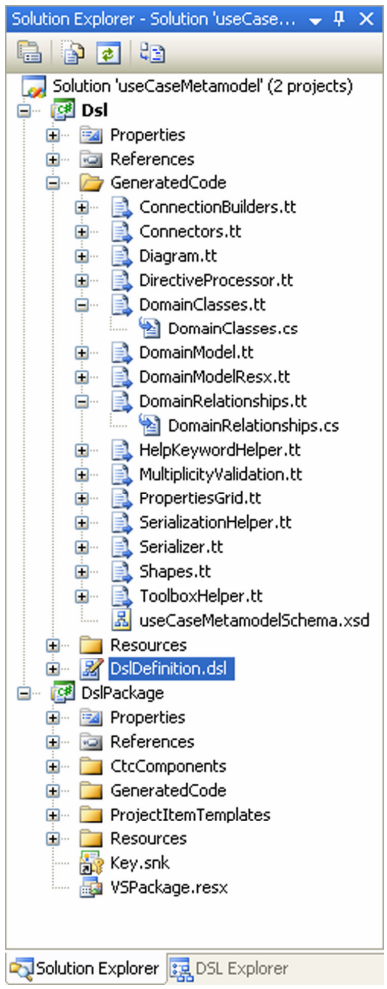

Figure 46 – File containing the definition of the DSL and files containing generated code

The elements that will be displayed in the *Experimental Designer*'s Toolbox are shown in DSL Explorer (see Figure 47). These are the notational elements available to build models in the *Experimental Designer*.

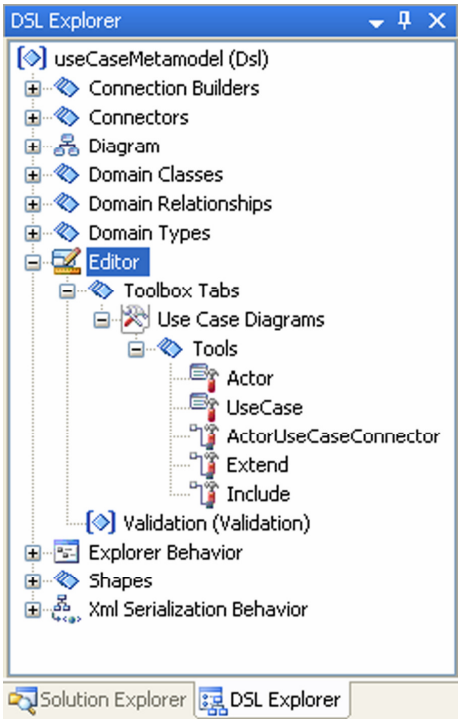

Figure 47 – Experimental Designer's Toolbox notational elements

For each *Experimental Designer* there is an associated diagram. Figure 48 illustrates that the use case metamodel has as diagram associated with the *Experimental Designer*, the diagram called *UseCaseDiagram*, which is the model to be built in the *Experimental Designer*.

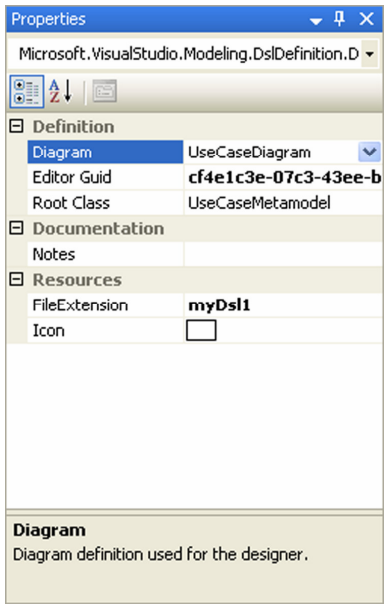

Figure 48 – Diagram associated with the Experimental Designer

Each diagram is represented by a class which is the root, at the metamodel's level, of all elements that will be part of the diagram. In the case presented in Figure 49, the class *UseCaseMetamodel* is the class being represented by the diagram *UseCaseDiagram*.

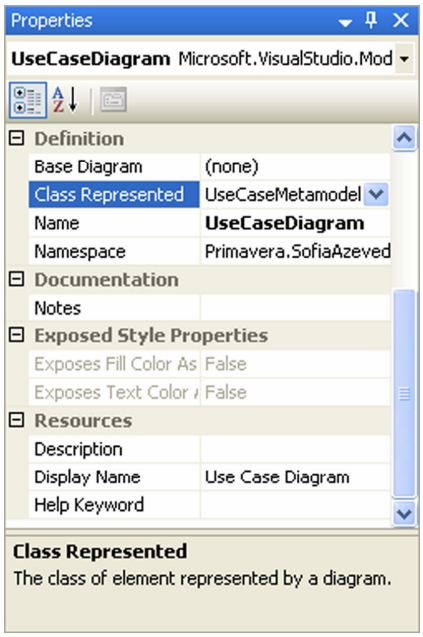

Figure 49 – Class represented by the diagram

The elements that can be concretized in the *Experimental Designer* have corresponding shapes. These shapes may have decorators. For instance, in the case of actors, their names must be displayed under the actor's figure, centred. That is precisely what the property *Position* of attribute *ActorName* defines, as we can see in Figure 50.

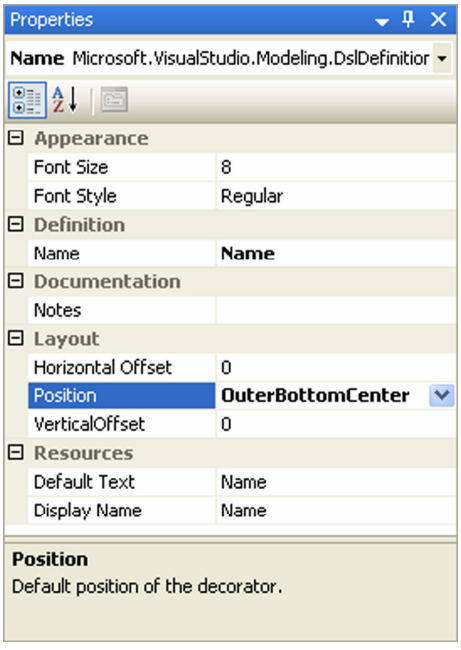

Figure 50 – Property Position

Besides the definition of the domain class and its corresponding shape, it is necessary to define the domain class' *Parent element path*, which corresponds to the path of the parent element of the domain class in the metamodel, as Figure 51 illustrates.

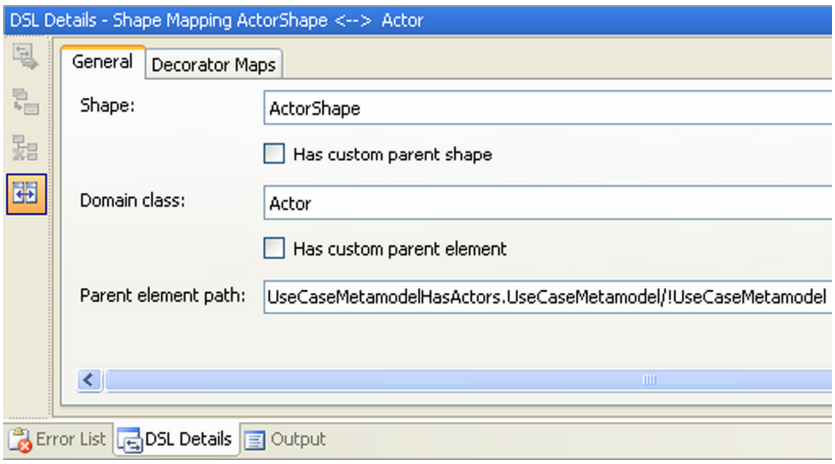

Figure 51 – Mapping of a domain class to a shape

A use case actor has, as mentioned previously, a decorator, which is the name of the actor. Figure 52 shows a drop-down menu where the attribute *Name* from the domain class called *Actor* can be chosen to be associated with the decorator in question.

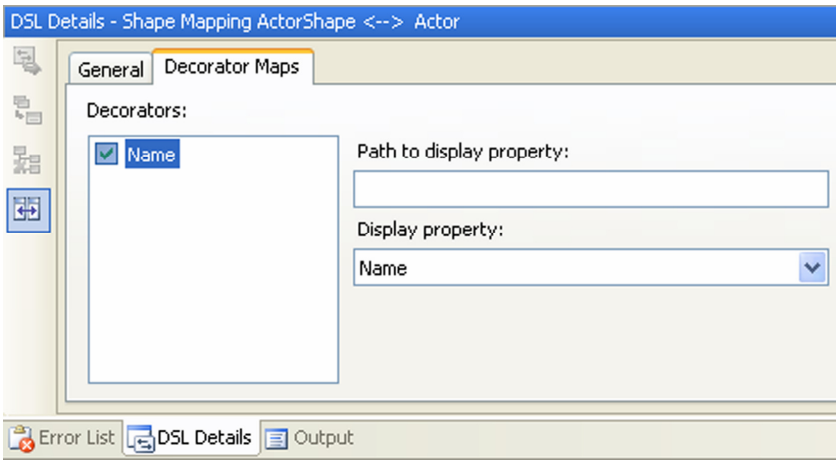

Figure 52 – Mapping of a domain class attribute to a decorator

Figure 53 depicts the *DSL Designer*'s Toolbox. The *DSL Designer* is the designer where DSLs (or other designers) can be defined. It is the designer that DSL Tools make available by default when a new project for a DSL designer is created.

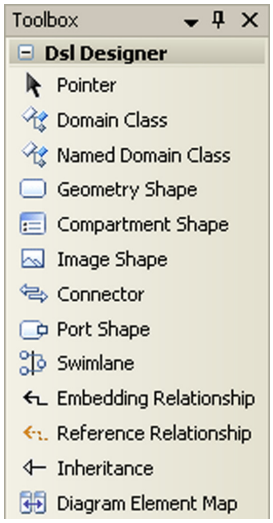

Figure 53 – DSL Designer's Toolbox

As to the elements shown in the Toolbox of the *Experimental Designer* (partially compiled from the *Designer*), Figure 54 illustrates the properties the element *Actor* has. That element represents the domain class *Actor*. The property *Caption* determines the name that will be displayed in the Toolbox associated with that same element. The property *Toolbox Icon* determines the symbol associated with the element in the Toolbox of the *Experimental Designer*.

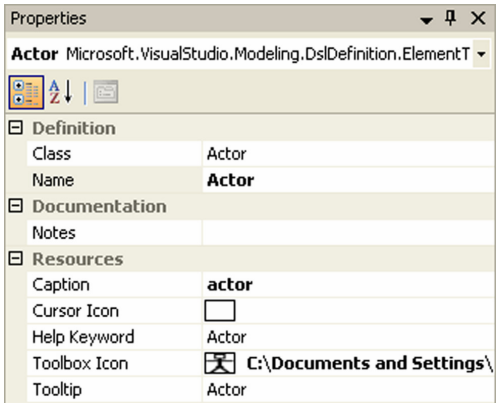

Figure 54 – Properties of an element from the Experimental Designer's Toolbox

The property *Inheritance Modifier* exists, ultimately, in order for the properties of a shape to need not to be defined for every shape sharing those properties. This property allows having an abstract domain class or an abstract domain relationship. Figure 55 depicts the property.

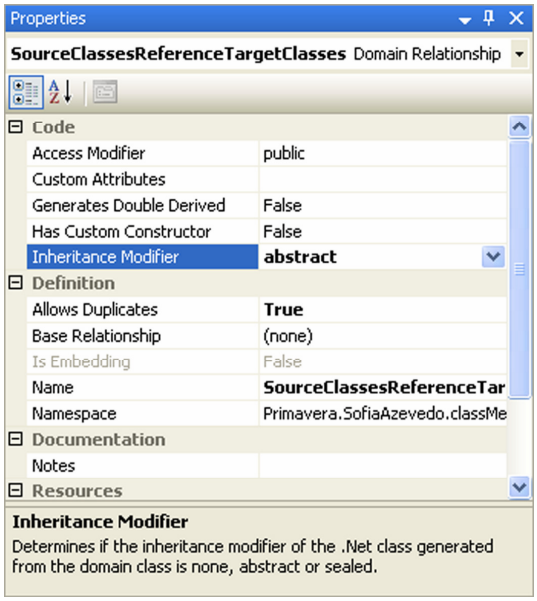

Figure 55 – Inheritance Modifier property

The property *Inheritance Modifier* implies that a base shape is defined for a shape, as illustrated in Figure 56.

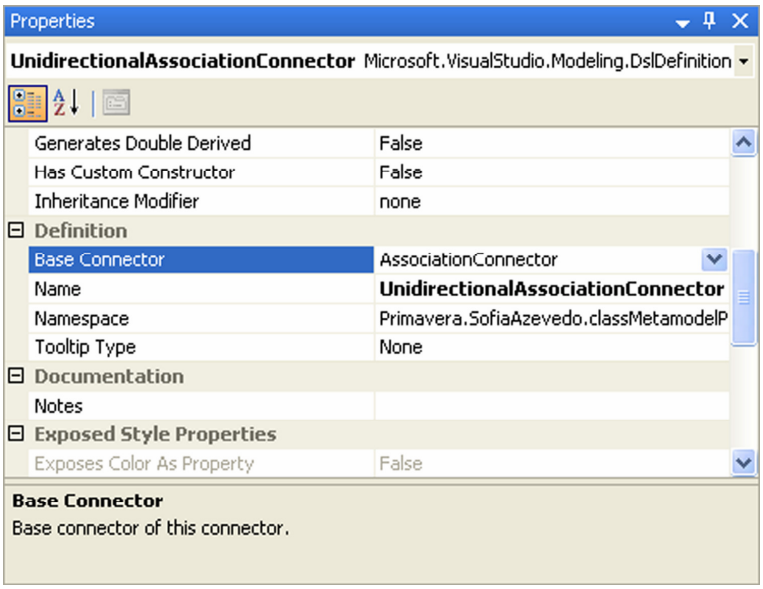

Figure 56 – Definition of a base shape for a connector

If the property called *Allows Duplicates* (shown in Figure 57) of the relationship *UseCaseMetamodelHasActors* takes the value *False*, then, there cannot be more than one relationship between the elements that can be connected through that relationship in the model.

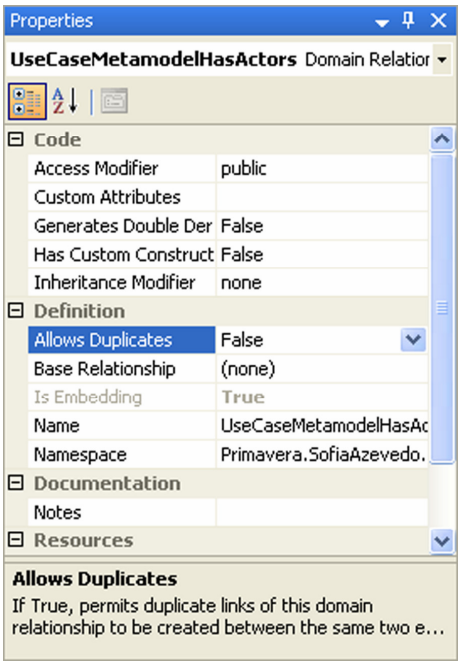

Figure 57 – Property Allows Duplicates