Generic Decorator Overview

Joe Porter, Tihamér Levendovzsky, Kevin Smyth, and Sandor Nyako
ISIS, Vanderbilt University

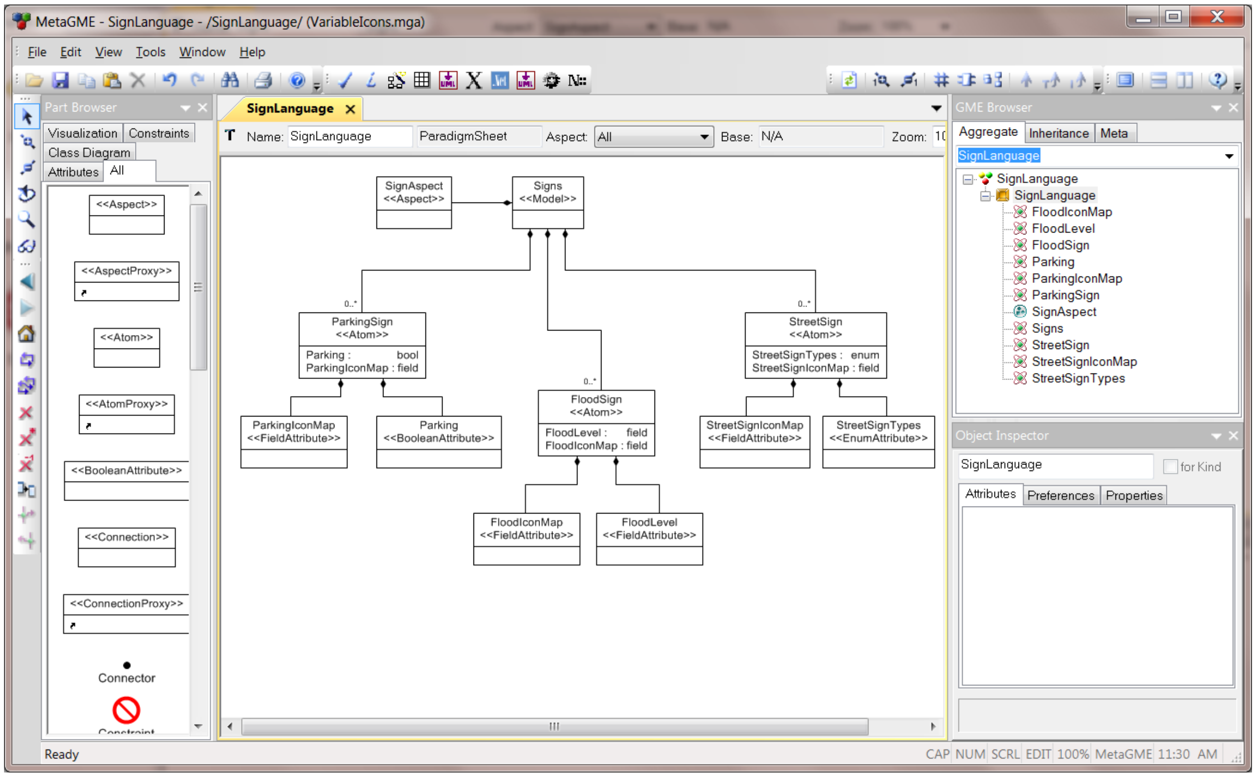
Problem

In GME, how do I display different icons for an object depending on the values of attributes within that component?

Solution

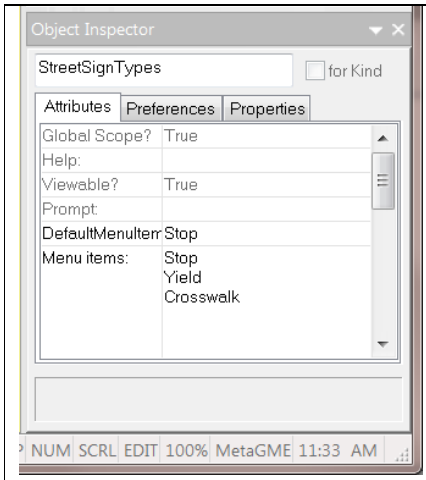
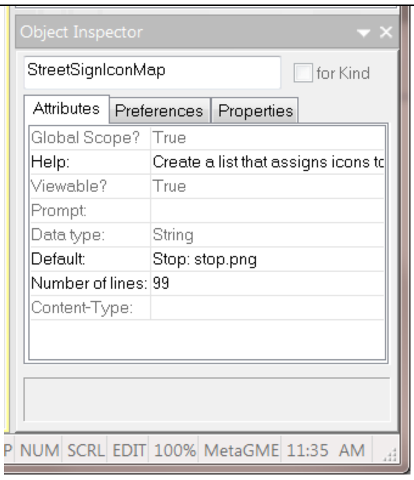
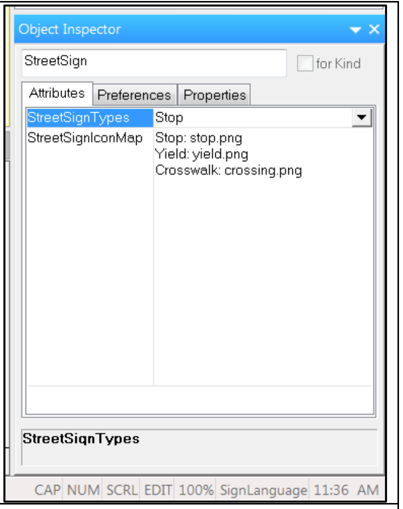
In GME, a decorator is a specialized object that provides the visual interface for components. We adapted an existing decorator (written in C#) to provide the basic functionality.

Example



We created a simple language called SignLanguage which models street signs. For each type of sign, we illustrate a different attribute type. The basic idea can be illustrated from the StreetSign class:

StreetSign has two attributes: StreetSignTypes and StreetSignIconMap. StreetSignTypes is an enumeration that allows the user to select between Stop, Yield, or Crosswalk signs as shown.

		
Metamodel detail for enumeration values in StreetSignTypes	Metamodel detail for multi-line string attribute that will define the icon map.	Model-level detail of the two attributes: the top drop-down allows the user to select the sign type, and the icon map strings assign different icon files to each option. Each of the three sign classes has an icon map.

Object Inspector
StreetSignTypes ☐ for Kind
Attributes | Preferences | Properties

Global Scope?	True
Help:	
Viewable?	True
Prompt:	
DefaultMenuItem	Stop
Menu items:	Stop Yield Crosswalk

NUM SCRL EDIT 100% MetaGME 11:33 AM

Object Inspector
StreetSignIconMap ☐ for Kind
Attributes | Preferences | Properties

Global Scope?	True
Help:	Create a list that assigns icons to
Viewable?	True
Prompt:	
Data type:	String
Default:	Stop: stop.png
Number of lines:	99
Content-Type:	

NUM SCRL EDIT 100% MetaGME 11:35 AM

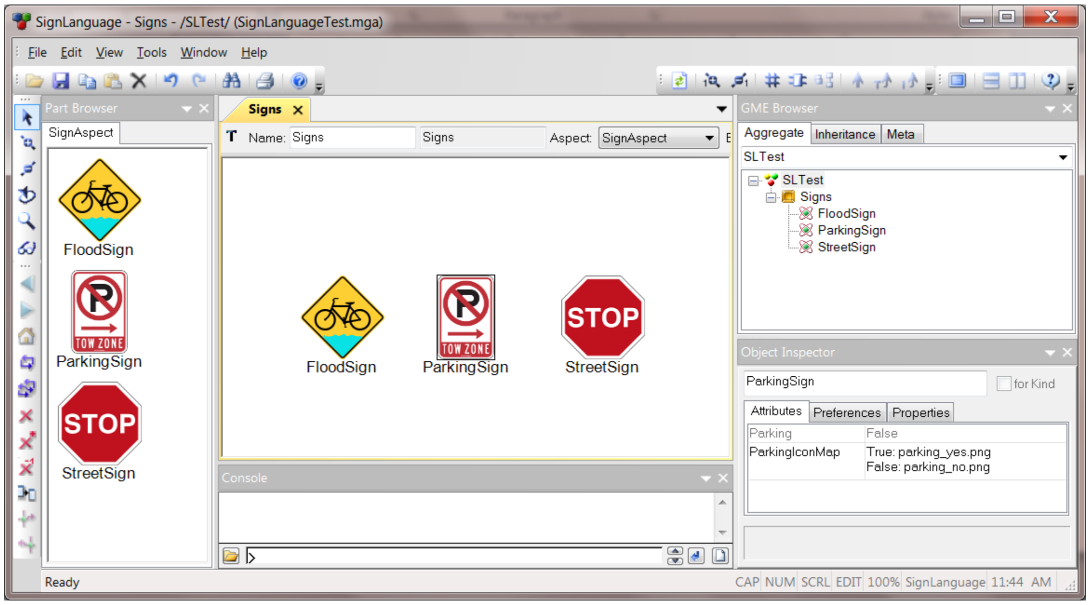
Object Inspector
StreetSign ☐ for Kind
Attributes | Preferences | Properties

StreetSignTypes	Stop
StreetSignIconMap	Stop: stop.png Yield: yield.png Crosswalk: crossing.png

StreetSignTypes

CAP NUM SCRL EDIT 100% SignLanguage 11:36 AM

The actual test model is shown below. Changing the relevant attribute changes the icon. We'll get into the details later.


SignLanguage - Signs - /SLTest/ (SignLanguageTest.mga)
File Edit View Tools Window Help
Part Browser
SignAspect
FloodSign
ParkingSign
StreetSign
Signs
FloodSign
ParkingSign
StreetSign
Console
GME Browser
Aggregate Inheritance Meta
Ready

Name: Signs   Signs   Aspect: SignAspect

SLTest
- SLTest
  - Signs
    - FloodSign
    - ParkingSign
    - StreetSign

Object Inspector
ParkingSign ☐ for Kind
Attributes | Preferences | Properties

Parking	False
ParkingIconMap	True: parking_yes.png False: parking_no.png

CAP NUM SCRL EDIT 100% SignLanguage 11:44 AM

Details

Generic Decorator File List

Infrastructure

- Properties/AssemblyInfo.cs – defines version numbers and other project-related constants
- GenericDecorator.csproj – Visual Studio 2010 Project File
- GenericDecorator.sln – Visual Studio 2010 Solution File

Decorator

- DecoratorBase.cs – very lowest level of the decorator inheritance hierarchy. Defines stubs for many of the decorator callbacks.
- DecoratorUIBase.cs – inherits from DecoratorBase, and defines some differentiating behavior (selected vs. not).
- DecoratorDefaultBitmapBase.cs – defines the DefaultBitmapDecorator class, which inherits from DecoratorUIBase. This class handles the selection and display of icon images, and embellishing those images based on whether the displayed object is an Instance, Reference, SubType, Type, or a Null reference. We modify this class to achieve the desired icon-changing effects.
- DefaultDecorator.cs – inherits from DefaultBitmapDecorator. This class adds the display of ports to the objects that can contain port objects.
- GenericDecorator.cs – Top level class that implements the COM interface to GME. The GenericDecorator instantiates an internal copy of DefaultDecorator, and then delegates events from the COM interface to the DefaultDecorator instance.

Language Files (in the meta/ subdirectory)

- SignLanguage.xme – example language definition (MetaGME)
- SignLanguageTest.xme – sample model file (SignLanguage)
- Icons/*.png – icon files for the language
- ISIS.GME.Dsml.SignLanguage.Interfaces.cs – domain-specific API generated from the MetaGME model (C#)
- ISIS.GME.Dsml.SignLanguage.Classes.cs – domain-specific API generated from the MetaGME model (part 2)
- ISIS.GME.Dsml.SignLanguage.dll – compiled version of the domain-specific API

Unused

BackgroundDraw.cs – the BackgroundDraw classes are not used in this example, but are left in the project in case examples are needed.

Code Description

In GME, a decorator is a class that draws an object. The decorator is instantiated by GME once for each object per aspect. When events are triggered for an object (e.g., move, change attribute value, etc…) the decorator is re-instantiated. In our example, the logic for selecting an icon occurs within the DefaultBitmapDecorator class constructor. We'll give a simple overview of that code, and then the icon selection code for each of the three object types. All of the code shown here comes from the file DecoratorDefaultBitmapBase.cs, as indicated above.

`public DefaultBitmapDecorator(GenericDecorator decorator,`
`            IntPtr parentHwnd, MgaFCO fco, MgaMetaFCO metafco,`
`            MgaProject project, IBackroundDraw backgrounddraw,`
`            Type formType = null)`
`          : base(parentHwnd, fco, project, formType != null)`
`      {`
	Constructor signature. fco is the object being redrawn.

`          string iconName;`
`          decorator.GetPreference(out iconName, "icon");`
`          string defaultIconName = iconName;`
	This code segment looks up the icon filename in the "standard" location, and saves it in case the logic fails to yield a valid filename.

`          if (fco != null)`
`          {`
`             string kind = fco.Meta.Name;`
`             if (kind == "ParkingSign")`
`             {`
`                iconName = GetParkingSignIcon(fco);`
`             }`
`             else if (kind == "FloodSign")`
`             {`
`                iconName = GetFloodSignIcon(fco);`
`             }`
`             else if (kind == "StreetSign")`
`             {`
`                iconName = GetStreetSignIcon(fco);`
`             }`
`             // else fall through to the default icon`
`          }`
`            …`
`      }`
	Based on the Meta class (type) of the object, use a different mechanism to get the icon filename. The code following this selection is omitted here, but it is available in the source package. After selection, the constructor code loads the icon file and determines how to embellish it.

Street Sign

StreetSign was displayed above. The icon is selected by looking up the value of the enumerated StreetSignTypes, and then matching the entry in the map.

```
    protected string GetStreetSignIcon(IMgaFCO ss)
    {
      string icon_map = ss.StrAttrByName["StreetSignIconMap"].Trim();
      string street_sign = ss.StrAttrByName["StreetSignTypes"].Trim();

      // expect list of form <key string> : <value string>\n
      var splitResult = icon_map.Split(new string[] { System.Environment.NewLine, "\n" },
        System.StringSplitOptions.RemoveEmptyEntries);

      foreach (string line in splitResult)
      {
        string[] halves = line.Split(new Char[] { ':' }, 2);
        d.Add(halves[0].Trim(), halves[1].Trim());
        if ( halves[0].Trim() == street_sign )
          return halves[1].Trim();
      }

      return ss.RegistryValue["icon"]; // default icon
    }
```

Parking Sign

The parking sign functions similarly to the street sign, but the selection variable is Boolean.

Icon map values:

True: parking_yes.png
False: parking_no.png

```
protected string GetParkingSignIcon(IMgaFCO ps)
    {
      string icon_map = ps.StrAttrByName["ParkingIconMap"].Trim();
      bool parking_val = ps.BoolAttrByName["Parking"];

      // expect list of form <key string> : <value string>
      var splitResult = icon_map.Split(new string[] { System.Environment.NewLine, "\n" },
        System.StringSplitOptions.RemoveEmptyEntries);

      foreach (string line in splitResult)
      {
        var halves = line.Split(new Char[] { ':' }, 2);
```

```
            var lhs_bool = Convert.ToBoolean(halves[0].Trim());

            if (parking_val == lhs_bool)
            {
                return halves[1].Trim();
            }
        }

        return ps.RegistryValue["icon"]; // default icon
    }
```

Flood Sign

The flood sign icon has a numeric value for the flood level. The map specifies the lower bound for the value that matches the correct icon. Negative numbers will be accepted but always map to the 0.0 entry.

0.0: bicycle_flood0.png (0.0 <= FloodLevel < 1.0)
1.0: bicycle_flood1.png (1.0 <= FloodLevel < 2.0)
2.0: bicycle_flood2.png (2.0 <= FloodLevel < 3.0)
3.0: bicycle_flood3.png (3.0 <= FloodLevel …)

```
    protected string GetFloodSignIcon(IMgaFCO fs)
    {
        string icon_map = fs.StrAttrByName["FloodIconMap"].Trim();
        double flood_level = fs.FloatAttrByName["FloodLevel"];

        // expect list of form <key string> : <value string>
        var splitResult = icon_map.Split(new string[] { System.Environment.NewLine, "\n" },
            System.StringSplitOptions.RemoveEmptyEntries);

        string iconStr = "";
        foreach (string line in splitResult.Reverse()) // go in decreasing order
        {
            var halves = line.Split(new Char[] { ':' }, 2);
            double level = Convert.ToDouble(halves[0].Trim());
            iconStr = halves[1].Trim();

            if (flood_level >= level )
                return iconStr;
        }

        return iconStr; // if flood_level is smaller than them all
    }
```