ASP.NET MVC - SampleApplication based on Entity Framework 4.0

The intention of the *MVCSampleApplication* was to build a sample application using the latest .NET technologies.

The ASP.NET MVC application provides a simple guestbook, where users can register and post comments. Administrators may additionally edit and delete comments. Moreover a WPF based client is included, using the same services as the web application.

This article describes the key implementation aspects and overall architecture of the application. It does not contain a step-by-step tutorial explaining every detail, as you find many blog posts discussing all the technologies being used by this project on the internet[1]. The goal here was to put all these technologies together in one application.

The following technologies are used:

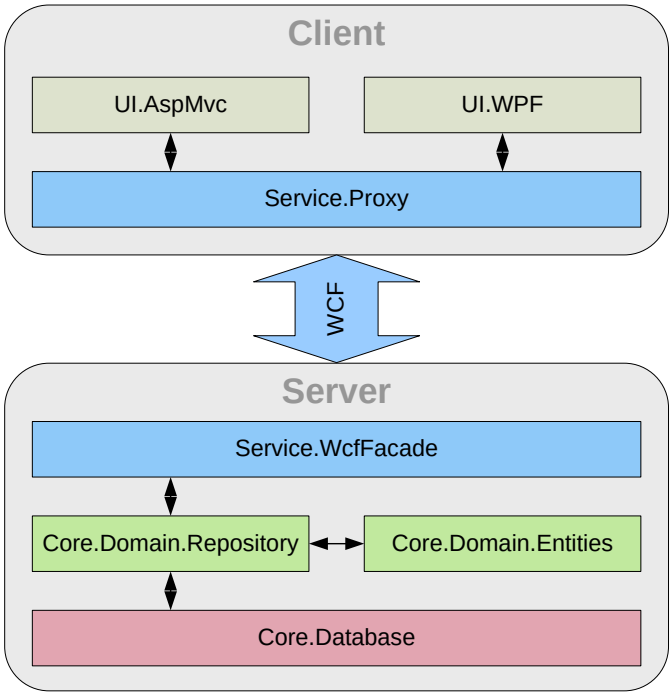
- UI
  - ASP.NET MVC with ELMAH and MVCSiteMap
  - WPF Client with WPFLocalizeExtension
- WCF service (both clients use this)
- Server
  - Entity Framework 4.0 with Self Tracking Entities

The following features and techniques are used:

- Fully localized application
- Dependency Injection using Unity
- Repository Pattern
- Unit of Work Pattern

[1] You will find some links at the end of this article.

Architecture



The Server

The server consists of four subprojects. The *Service.WcfFacade* contains the service interface published as WCF service. The implementation of the service interface relies on *Core.Domain.Repository*.

Core.Domain.Repository only contains some helper classes and interfaces which a concrete repository must implement. Different implementations of this interface are imaginable, using nHibernate or Linq2SQL for example. In *Core.Database* the repository is implemented on basis of the new Entity Framework 4.0 together with a SQL-Express database.

In the past the Entity Framework did not work seamlessly with WCF services. Reattaching updated objects to the context always required a workaround. One possibility was to send both, the original object and the modified copy and then redo the changes on the server. Another workaround was to manually set all properties as modified.

With the new Entity Framework you'll get more options. You are now able to use custom T4 templates to generate your entity classes or you could write your entities manually.

With the Microsoft ADO.NET Entity Framework Feature CTP 1 you will get templates for:

- POCO (Plain Old CLR Objects) Entities
- Self-Tracking Entities (N-Tier support)

Since this project uses WCF, the template for Self-Tracking Entities is utilized[2].

[2] http://blogs.msdn.com/adonet/archive/2009/06/22/feature-ctp-walkthrough-self-tracking-entities-for-entity-framework.aspx

The Clients

Both clients rely on *Service.Proxy*, which contains the service reference to the WCF service. The advantage of placing the reference in a separate project is, that the proxy classes have only to be generated once, when the service interface changes.

WCF Localization

Localizing an application is not very difficult. If you use *.resx-files for your strings, you have already accomplished a lot of work. You only have to ensure that the thread accessing a resource has the right culture applied.

If you have a client-server system, the server does not know about the language in the client, so you must provide the language in every call, since multiple clients running in different languages may access the server.

One option is to add an additional argument to every service method to pass the current client language to the server. A better approach is to use a WCF behavior to pass the language automatically.

You can use `MessageInspectors` to add/read the current language as string to/from every message header. This can be done in the following way:

- Create a class that implements the interfaces `IClientMessageInspector` and `IDispatchMessageInspector`. You only have to implement the following methods:
  - `object BeforeSendRequest(ref Message request, IClientChannel channel)`
    This method adds the current language as string to the message header.
  - `object AfterReceiveRequest(ref Message request, IClientChannel channel, InstanceContext instanceContext)`
    This method reads the current language from the message header on the server side.
- Create a class that implements the interface `IEndpointBehavior` and/or `IServiceBehavior`. This class adds an instance of the `MessageInspector` to every endpoint/service.
- Configure the behavior in your `Web.config` file. Therefore you have to create a class that derives from `BehaviorExtensionElement`. This class simply works as a factory for the behavior.

If you want to see the full implementation and configuration take a look at:

- `src\Service.WcfFacade\I18n`
- `src\UI.AspMvc\Web.config`

Dependency Injection

Unity is used as Dependency Injection container. You can either configure the container in code or over the `Web.config` file. If you configure the container in code, you yet compiler checks and you can easily correct errors, but you need references to every assembly you use.

Since I wanted to avoid unnecessary project references, I chose to use the `Web.config` for configuration. The project *Common* contains a helper class which reads the configuration and resolves instances of your interfaces.

The Web Frontend (ASP.NET MVC)

As primary frontend an ASP.NET MVC application is used. This application also hosts the WCF service which is required by both, the web and the WPF client.

If you want to learn ASP.NET MVC I recommend the book ASP .NET MVC 1.0[3] to you.

The *MVCSampleApplication* is not very extensive and quite easy to understand. It consists of 4 controllers and some views.

Localization

All controllers derive from `LocalizedControllerBase` which manages the different languages of the page. The default language is English, as a second language German is available. If a user visits the website for the first time, the preferred language is read from the `HttpContext`, if the language is available, it is stored in the session. The visitor may also change the language manually, simply by clicking on the corresponding flag, which triggers a `GET` request with a certain query string:

The `LocalizedControllerBase` will recognize the language in the query string and apply it. In order to get this working on all pages, all controllers must derive from the mentioned class.

On any further request the language from the session will be used, unless it is changed manually again.

The views and controllers retrieve their resources from *.`resx`-files. Since controllers are not associated with a special view, they could only use global resources. Some extension methods enable you to easily access the resources within views and controllers[4]. In `Global.asax` an extended `ViewEngine` is registered to enable localization.

Error Logging

To log unhandled exceptions I use ELMAH. The wiki on the project's website explains everything you need to know for configuration. One problem is that ELMAH only logs unhandled exception. When you use the `HandleErrorAttribute` on your controllers, no errors will get logged. There are several solutions to this problem; one is to override the `HandleErrorAttribute`[5].

MVC T4 Template

Normally you have to use strings to address your controller actions, e.g.

```
<%= Html.ActionLink("Back to index", "Index") %>
```
(In a view)

```
return RedirectToAction("Index");
```
(In a controller)

---

[3] The first chapter can be downloaded here: http://www.hanselman.com/blog/FreeASPNETMVCEBookNerdDinnercomWalkthrough.aspx

[4] http://blog.eworldui.net/post/2008/10/ASPNET-MVC-Simplified-Localization-via-ViewEngines.aspx

[5] http://stackoverflow.com/questions/766610/how-to-get-elmah-to-work-with-asp-net-mvc-handleerror-attribute/779961

You will not get a complier error if you rename a controller action. If you use the MVC T4 template written by David Ebbo you will.

The template analyzes your controllers and generates some code for you. You can rewrite the above examples the following way:

```
<%= Html.ActionLink("Back to index", MVC.Home.Index())%>
```
(In a view)

```
return RedirectToAction(MVC.Home.Index());
```
(In a controller)

SiteMap

Since the classic SiteMap from ASP.NET does not work in a MVC context, you can use the MVCSiteMap provider.

TODOs

There are still some tasks to do, before you should use this application in a production environment:

- Unit Tests have to be written
- Write (more) logging output using log4net
- User passwords must be encrypted
- WCF service must be secured

Technologies / Libraries

Visual Studio 2010

ASP.NET MVC 2.0

Microsoft ADO.NET Entity Framework Feature CTP 1

StyleCop

http://code.google.com/p/elmah

http://unity.codeplex.com

http://mvcsitemap.codeplex.com

http://wpflocalizeextension.codeplex.com

http://logging.apache.org/log4net/index.html

Resources

http://blogs.msdn.com/adonet/archive/2009/05/21/poco-in-the-entity-framework-part-1-the-experience.aspx

http://blogs.msdn.com/adonet/archive/2009/05/28/poco-in-the-entity-framework-part-2-complex-types-deferred-loading-and-explicit-loading.aspx

http://blogs.msdn.com/adonet/archive/2009/06/22/feature-ctp-walkthrough-poco-templates-for-entity-framework.aspx

http://blogs.msdn.com/adonet/archive/2009/06/22/feature-ctp-walkthrough-self-tracking-entities-for-entity-framework.aspx

http://blog.keithpatton.com/2009/05/30/Entity+Framework+POCO+Repository+Using+Visual+Studio+2010+Net+40+Beta+1.aspx

http://devtalk.dk/2009/06/09/Entity+Framework+40+Beta+1+POCO+ObjectSet+Repository+And+UnitOfWork.aspx

http://www.hanselman.com/blog/FreeASPNETMVCEBookNerdDinnercomWalkthrough.aspx

http://blog.eworldui.net/post/2008/10/ASPNET-MVC-Simplified-Localization-via-ViewEngines.aspx

http://blogs.msdn.com/davidebb/archive/2009/06/17/a-new-and-improved-asp-net-mvc-t4-template.aspx

http://stackoverflow.com/questions/766610/how-to-get-elmah-to-work-with-asp-net-mvc-handleerror-attribute/779961