Basisontwerp Externe connector

beschrijving connectoren

Leo Broos

2-7-2011

Dit document beschrijft het ontwerp op hoofdlijnen en abstractie voor het bouwen van connectoren naar externe systemen vanuit een software design view.

Inhoudsopgave

1. De componenten

De volgende componenten kunnen worden herkend om een connector te definiëren die los staat van de software die het gaat implementeren. Om een zo hoog mogelijk niveau van “Lose coupling” te bereiken zijn deze componenten minimaal noodzakelijk.

1.1. De Interface

Om een losse koppeling te kunnen garanderen, moet de functionaliteit die de connector implementeert losgekoppeld zijn door een interface. Deze interface beschrijft eigenschappen (properties) en methoden zonder hun eigenlijke implementatie. Tijdens runtime wordt bepaalt waar de implementatie plaatsvindt, waardoor de connector een blackbox wordt met alleen in/output.

1.2. De Serviceagent

In de Serviceagent vindt de feitelijke implementatie plaats van waaruit bericht ontvangen en verstuurd worden en eventuele validaties en conversie om de berichten in de juiste vorm c.q. objecten door te geven. De serviceagent maakt bij Webservices gebruik van de gegenereerde proxyklassen op basis van de WSDL’s van de Webservices en vormt de correcte XML m.b.v. de/serialisatie van objecten. Het .NET Framework ondersteund dit principe volledig en biedt de XML namespace in combinatie met LINQ om hier optimaal gebruik van te maken.

1.3. De Proxy klassen

De proxy klassen worden gegenereerd door een referentie leggen naar een Webservice binnen Visual Studio in de SoluitonExplorer. Deze verzorgen de feitelijk communicatie naar de buitenwereld toe volgens SOAP.

1.4. De Domeinobjecten

Om de communicatie plaats te laten vinden d.m.v. objecten die gede-/serialiseerd worden zullen deze hier in aparte klassen gedefinieerd moeten worden. Door de XML namespace tags te gebruiken kun je grote controle verkrijgen over de serialisatie van de objecten. Verder moet per geval bekeken worden of bestaande domeinobjecten hergebruikt kunnen worden en/of de objecten uit de proxy klassen gebruikt kunnen worden. E.e.a. is ook afhankelijk van het type Webservice dat aangeroepen wordt, sommige gebruiken objecten (WCF services) en andere alleen XML in de vorm van een String meegestuurd wordt (ASMX services).

1.5. Validatie klassen

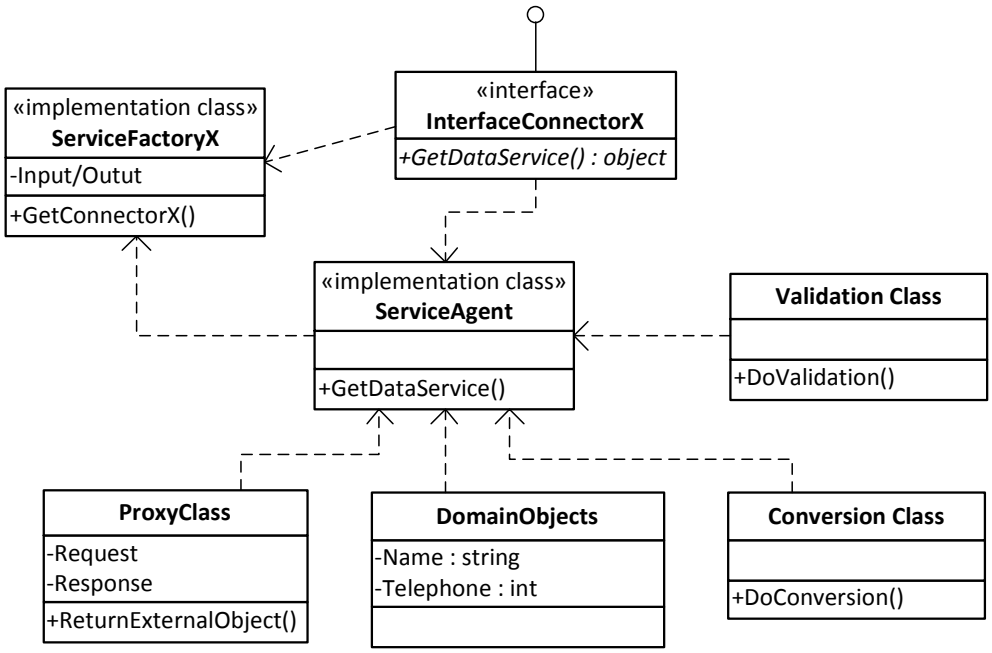
Het is altijd gewenst om bepaalde validaties zoals type conversies zo dicht mogelijk tegen de bron te doen. Daarom is het gewenst om validatie hier uit te voeren zowel voor in als uitgaande berichten.

1.6. Geen Business logic

Business logic houden we bewust buiten de connector, om zo zeker te zijn dat ‘n losse koppeling gegarandeerd blijft. Deze logica moet gebeurd zijn alvorens de connector aangesproken wordt, dus in software die de connector gebruikt of de service waarmee de connector communiceert.

2. Positionering van de componenten in de connector

Onderstaande afbeelding geeft een overzicht van de positionering van de componenten binnen de connector en de interactie van de connector met de buitenwereld.

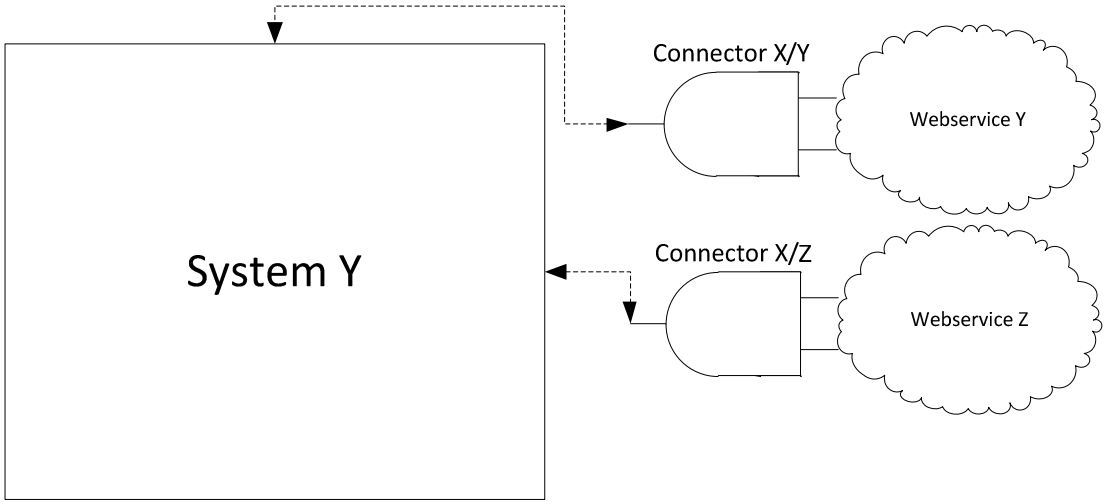


Deze afbeelding toont dat de Interface als de connectie naar het systeem en de proxy klasse als de connectie met een Webservice. De feitelijke implementatie gebeurd in de Service-Agent, die gebruik maakt van validatie, conversie en eventuele Domein Objecten om de berichten door te geven. Zie ook de beschrijving onder 1. Componenten. Een extra optionele klasse die ik graag gebruik om Polymorphisme te bewerkstelligen is de ServiceFactory klasse, deze geeft een instantie van de Interface terug en op deze wijze verbergt het de fysieke implementatie in de serviceagent.

3. Positionering van de connector in de architectuur

Hierna wordt een voorbeeld architectuur weergegeven in een n-tier omgeving, waarbij de connector duidelijk als een extern component wordt gepositioneerd. Uiteraard moet de connector kennis hebben van beiden systemen qua object/domein modellen, echter niet meer dan nodig is om te kunnen communiceren tussen de interface naar de proxy en visa versa. Dit maakt hergebruik van de connector mogelijk. In praktijk betekent dit dat de Domeinobjecten de worden gebruikt vanuit de Proxy klasse worden gedicteerd en geïnstantieerd vanuit het aanroepende systeem door de Interface. Immers als kennis gedeeld wordt over het interne object vanuit het implementerende

systeem is hergebruik door een ander systeem onmogelijk, zonder aanpassing van de connector intern.



Hiermee wordt uitgebeeld dat de connector de plug c.q. stekker is waarmee de verbinding naar een extern systeem word gemaakt. In het voorbeeld een n-tier omgeving met Webservices, maar deze kan evenzo goed services bevatten die op een fysieke server draaien. Het verbindingsprotocol wordt bepaald door het externe systeem (Webservices Y, Z in casus), maar geïmplementeerd in de connector, hierdoor hoeft het systeem wat de connector gebruikt geen kennis te hebben van gebruikte protocollen.

Deze kennis impliceert automatisch dat inzet van WCF technologie hierbij een groot voordeel kan opleveren om hergebruik te vergemakkelijken. WCF stelt de developer in staat om pas op het laatste moment een keuze te hoeven maken voor het te gebruiken protocol naar het extern systeem c.q. aanpassing voor een ander protocol mogelijk maakt met geen of minimale aanpassing van de software zelf.

4. Hoe ziet dit er in de praktijk uit?

Nu is het tijd om een praktisch voorbeeld te geven, de .NET uitwerking is terug te vinden in de bijlagen die bij deze beschrijving horen.

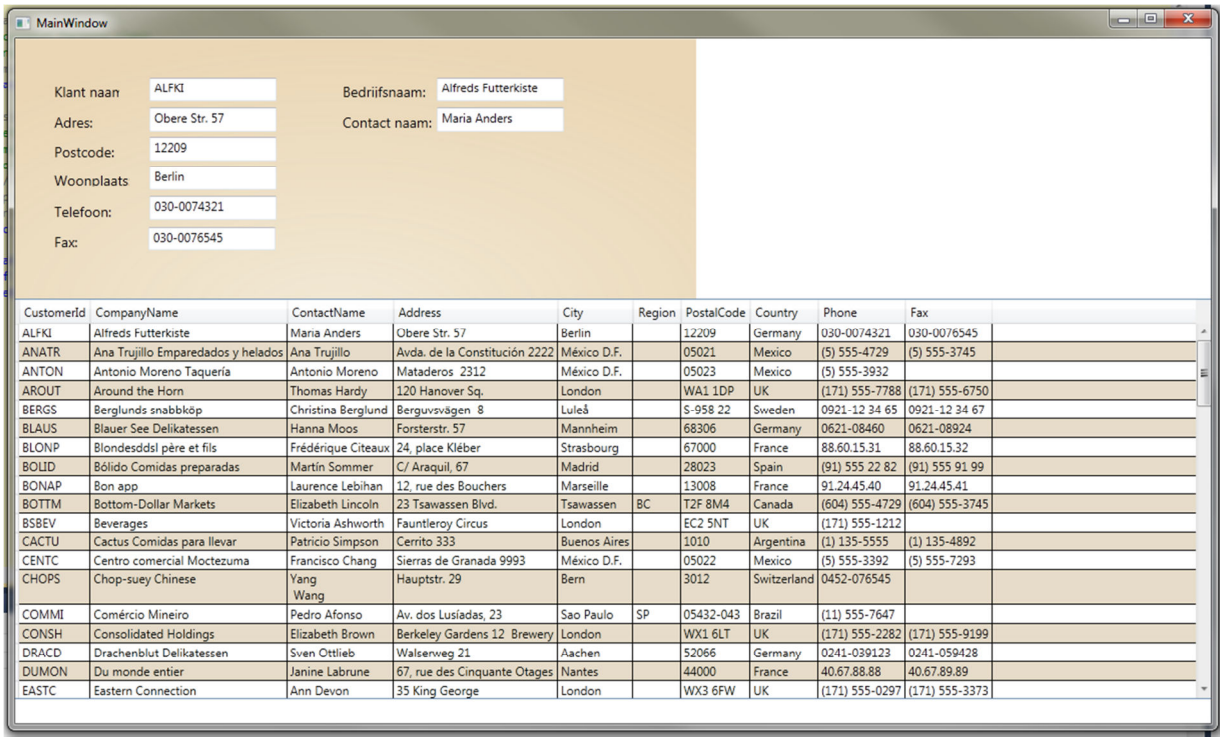
We gaan uit van 2 Client applicaties, 2 Webservices en 2 connectoren, dit is voldoende om aan te tonen dat de we de connectoren uit kunnen wisselen zonder deze intern aan te hoeven passen. De Client applicaties hoeven alleen een instantie van de interface van de connectoren aan te roepen.

4.1. De Client applicaties

Voor het gemak noemen we de Client applicaties Application A en B, om een beetje diversiteit te houden zijn ze uitgewerkt in een WindowsForm en een WPF applicatie.

Beiden applicaties zijn opgebouwd in een Master Detail concept, dus een grid in combinatie met detail informatie. Om het interessant te houden is de grid-informatie afkomstig van een andere Webservice dan de detail informatie. In praktijk betekent dit dat er een primaire sleutel zou moeten zijn om de informatie in de tweede Webservice op te kunnen halen in casus de CustomerID.

Hieronder volgt een afbeelding van de WPF applicatie.



CustomerId	CompanyName	ContactName	Address	City	Region	PostalCode	Country	Phone	Fax
ALFKI	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin		12209	Germany	030-0074321	030-0076545
ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.		05021	Mexico	(5) 555-4729	(5) 555-3745
ANTON	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.		05023	Mexico	(5) 555-3932	
AROUT	Around the Horn	Thomas Hardy	120 Hanover Sq.	London		WA1 1DP	UK	(171) 555-7788	(171) 555-6750
BERGS	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå		S-958 22	Sweden	0921-12 34 65	0921-12 34 67
BLAUS	Blauer See Delikatessen	Hanna Moos	Forsterstr. 57	Mannheim		68306	Germany	0621-08460	0621-08924
BLONP	Blondesddsl père et fils	Frédérique Citeaux	24, place Kléber	Strasbourg		67000	France	88.60.15.31	88.60.15.32
BOLID	Bólido Comidas preparadas	Martín Sommer	C/ Araquil, 67	Madrid		28023	Spain	(91) 555 22 82	(91) 555 91 99
BONAP	Bon app	Laurence Lebihan	12, rue des Bouchers	Marseille		13008	France	91.24.45.40	91.24.45.41
BOTTM	Bottom-Dollar Markets	Elizabeth Lincoln	23 Tsawassen Blvd.	Tsawassen	BC	T2F 8M4	Canada	(604) 555-4729	(604) 555-3745
BSBEV	Beverages	Victoria Ashworth	Fauntleroy Circus	London		EC2 5NT	UK	(171) 555-1212	
CACTU	Cactus Comidas para llevar	Patricio Simpson	Cerrito 333	Buenos Aires		1010	Argentina	(1) 135-5555	(1) 135-4892
CENTC	Centro comercial Moctezuma	Francisco Chang	Sierras de Granada 9993	México D.F.		05022	Mexico	(5) 555-3392	(5) 555-7293
CHOPS	Chop-suey Chinese	Yang Wang	Hauptstr. 29	Bern		3012	Switzerland	0452-076545	
COMMI	Comércio Mineiro	Pedro Afonso	Av. dos Lusíadas, 23	Sao Paulo	SP	05432-043	Brazil	(11) 555-7647	
CONSH	Consolidated Holdings	Elizabeth Brown	Berkeley Gardens 12 Brewery	London		WX1 6LT	UK	(171) 555-2282	(171) 555-9199
DRACD	Drachenblut Delikatessen	Sven Ottlieb	Walserweg 21	Aachen		52066	Germany	0241-039123	0241-059428
DUMON	Du monde entier	Janine Labrune	67, rue des Cinquante Otages	Nantes		44000	France	40.67.88.88	40.67.89.89
EASTC	Eastern Connection	Ann Devon	35 King George	London		WX3 6FW	UK	(171) 555-0297	(171) 555-3373

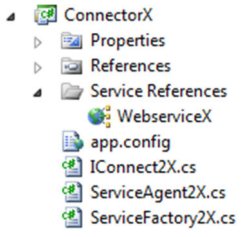
Beiden clients zijn in een MVC pattern opgebouwd, met aparte views en een bijbehorende controller, gezien de kleine omvang de applicaties. MVC staat voor Model View Controller, voor meer informatie hierover verwijs ik naar het internet, sleutelwoorden MVC en “Gang of four”.
Indien een regel geselecteerd wordt in het grid, zal de bijbehorende detailinformatie opgehaald worden uit de ander Webservice. Dit wordt geregeld via de Controller hierin wordt een instantie via de servicefactory van de dataconnector aangeroepen in de Get methoden en het resultaat in een property gestoken, behalve bij de Windows client, hier worden de Customers die direct gebind tegen het datagridView. Hiermee gelijk demonstrerend dat object/property binding tegen controls met WPF een stuk gemakkelijker gaat.

4.2. De Webservices

Voor de variatie een Webservice met WCF en een traditionele ASP.NET Webservice gemaakt, de WCF versie geeft de grid informatie en de andere Webservice de detail informatie. Voor beiden is een connector gemaakt die binnen de Client applicatie uitwisselbaar is door implementatie van de desbetreffende connector interface. Zie de voorbeeld code. Opmerking: de Webservices worden gehost binnen Visual Studio, dus dit is een prerequisite om het voorbeeld te kunnen draaien. Anders zult u genoodzaakt zijn om de Webservices zelf binnen IIS te configureren. In principe zijn het Stubs die een XML file inlezen, om zo zonder afhankelijkheid van een database te kunnen draaien.

4.3. De Connectoren

Zoals je in de bijbehorende code kunt zien is de connector exact opgebouwd zoals beschreven:



Interface, servicefactory en serviceagent.
De interface beschrijft het contract, de serviceagent implementeert deze en de servicefactory geeft een instantie van de interface om de gewenste functionaliteit aan te kunnen roepen. Zonder de interne werking bloot te geven (Polymorphisme).

De werking is geïncapsuleerd en er is (her)gebruik gemaakt van een en hetzelfde domein model, dit geeft ook gelijk een verschil weer tussen WCF en traditionele Webservices. Zoals je in code kunt zien implementeert en communiceert de WCF service het correcte domeinmodel, terwijl de asmx (klassieke) webservice een verborgen serialisatie heeft gedaan. Het Customer object bestaat hierdoor uit properties met Nederlandse namen zoals in de Xml tags is beschreven. Zie hieronder:

```
[XmlElement("Bedrijfsnaam", typeof(string))]
public string CompanyName { get; set; }
```

Een ander bijkomend voordeel is de testbaarheid van de connectoren. Door de losse koppeling kan de werking zowel geheel los (unittest) met stubs (integratietest) alsmede tegen de werkelijke services (acceptatietest) getest worden. Waarbij dit geheel geautomatiseerd kan gebeuren, al dan niet met gebruik van de Visual Studio testsuite en bijbehorende WCF client configuratie file(s).

4.4. De tests

In de voorbeeldcode zijn unittests uitgewerkt met een dekkingsgraad van 100% voor connector X en meer dan 90% voor Y. Het verschil zit in het type Webservice, voor de WCF service is een meer zuivere proxy gegenereerd, terwijl de proxy voor de asmx service meer overhead heeft die niet interessant is om te testen.

Deze unittests zijn geen zuivere testen per unit, de (Mock) Webservice moet draaien in de Visual Studio Proxy server of op IIS. De Url's naar de services moet kloppen met je lokale instanties. Om deze redenen is het meer een integratie test dan een unit test, maar het geeft wel een goed beeld van de werking. Als de Mock services opgezet zijn kunnen de geautomatiseerde tests in een daily build. Een andere methodiek voor Service Mocking geeft SoapUI, zie ook:
http://www.soapui.org/Service-Mocking/mocking-soap-services.html

In de Instance testen worden de connectoren geïnstantieerd en de bijbehorende methode om resultaat op te halen, aangeroepen. Waarna dit resultaat gecontroleerd wordt.

Daarna worden tests uitgevoerd om de verschillende constructor overloads van de gegenereerde proxy klassen te instantiëren en een resultaat aan te roepen, waarna dit resultaat gecontroleerd wordt.

4.5. Onderhoudbaarheid/Maintainability

Indien de Webservice verandert zal ook de connector aangepast moeten worden, inclusief de testen. Echter de applicatie die ze gebruikt niet, immers de Interface wordt alleen uitgebreid en de consumerende applicatie instantieert de Interface echt ter implementeert deze niet. Wordt de connector uitgebreid, dan moet er de veranderde Interface opnieuw worden geïmplementeerd in de serviceagent, zodat het bestaande contract niet wordt verandert en nogmaals, er geen aanpassingen nodig zijn in de consumerende applicaties. Tenzij er besloten wordt dat alle consumerende

applicaties een uitbreiding moeten krijgen. In dat geval hoeft alleen de nieuwe functionaliteit na instantiering van de Interface, via de servicefactory, aangeroepen te worden.

Voorbeeld: ConnectorX heeft Interface `IConnect2X` als contract met na instantiering de methode "`GetCustomer(string customerID)`" voor het ophalen van een specifieke customer. Stel nu dat er aan de interface een methode toegevoegd wordt "`GetOrdersByCustomer(string customerID)`" die alle orders teruggeeft voor een specifieke klant op basis van zijn customerID. Dit betekent dat de internal toegankelijke serviceagent een implementatie geeft voor deze methode en de aanroep hiervan geëncapsuleerd wordt door de servicefactory.

Oftewel na de instantiering binnen de consumerende applicaties, zoals in de MasterDetailController van ApplicationB, er een instantie gemaakt wordt zoals hieronder:

```
IConnect2X iserviceX = ServiceFactory2X.GetConnector2X(UrlX);
```

De iservicex heeft dan de mogelijkheid om GetOrdersByCustomer() aan te roepen en geeft de orders terug. Om deze orders in geheugen bij te houden moet het Domain uitgebreid worden met de entiteit c.q. het object Orders. In de controller klasse wordt dan een property van dit type bijgehouden waarna het resultaat weer ter beschikking staat van de GUI.

Bijvoorbeeld een tekst property van een Control kan dan weer als volgt gebonden worden aan een object property:

```
Text="{Binding Path=Orders.OrderId, Mode=TwoWay}"
```

En een collectie Control, zoals een List of DataGrid, kan zijn ItemsSource property binden als volgt:

```
ItemsSource="{Binding Path=Orders, Mode=TwoWay}"
```

Hiermee hebben we ook gelijk een verbinding gemaakt tussen een relationeel datamodel en entiteitenmodel. Waarbij WPF dan weer gemakkelijker de binding tussen het entiteitenmodel en de Controls verzorgt, door implementatie van het MVC patroon (of MVVM).