Jeroen Huitink
werkt bij Warp11, een onderdeel van Cap Gemini Ernst & Young. Hij is daar onder meer actief als senior Software- & Infrastructure Engineer. Jeroen Huitink was lid van het Early Adaptor Team van Cap Gemini Ernst & Young (zie http://www.jaggle.com) en heeft inmiddels ruim 1½ jaar ervaring met Microsoft .NET.

.NET en integratie met andere platformen

SYSTEEMKOPPELINGEN VIA WEBSERVICES

Met het op de markt komen van Microsoft .NET is een nieuwe fase aangebroken voor het ontwikkelen van software. Met de manier van denken die achter .NET schuilgaat en de nieuwe ontwikkelomgeving wordt een forse sprong gemaakt in de productiviteit van de programmeur.

Maar er is meer in de wereld dan alleen Microsoft .NET. Integratie met andere pakketten en omgevingen is vaak een must om de gebouwde software in bestaande organisaties in te bedden, of om te kunnen communiceren met andere organisaties. In dit artikel wordt ingegaan op de architectuurkeuzes die moeten worden gemaakt bij het bouwen van een koppeling, en laat zien op welke wijze deze koppeling vervolgens daadwerkelijk tot stand kan komen.

Over integratie

De integratie van applicaties en systemen is de grootste uitdaging die we nu kennen en dit zal in de toekomst voorlopig nog wel het geval zijn. Veel bedrijven hebben ondertussen automatiseringssoftware, al dan niet zelf gebouwd, om de kernactiviteiten van het bedrijf uit te voeren. Het is ondenkbaar om zonder deze software te werken, en dus moet bij het bouwen van een nieuwe applicatie rekening worden gehouden met de bestaande omgeving. Of in het geval van bijvoorbeeld een fusie tussen twee bedrijven dienen bestaande systemen te worden geïntegreerd.

Een goede integratie kenmerkt zich door openheid en duidelijkheid. Bij het bouwen van een koppeling om de integratie te bewerkstelligen, moet rekening worden gehouden met de volgende zaken:

- Openheid van de koppeling. Het is een koppeling tussen twee applicaties, dus is een protocol belangrijk. Hoe communiceren de applicaties met elkaar? Dit protocol moet bij voorkeur open en inzichtelijk zijn. Door een koppeling strak te implementeren kan deze weliswaar snel zijn, de geïntegreerde systemen zijn op deze manier echter wel onlosmakelijk aan elkaar verbonden (en zodoende één systeem geworden). Een vervanging van een van de twee systemen zal in dat geval een grote impact hebben op de koppeling.
- Een tweede belangrijk kenmerk is dat de koppeling op de juiste manier is gelegd. Een goed voorbeeld hiervoor is de noodzaak tot actualiteit. Een offline koppeling[1] kan alleen dan als de in de applicatie aanwezige gegevens niet tot op de minuut actueel hoeven te zijn.
- Zekerheid van de koppeling. Hoewel de koppeling een open karakter moet hebben, moet deze wel zeker zijn. Denk hierbij aan zaken als beveiliging en transactioneel gedrag.
- Gebruik het geëigende mechanisme. Als een systeem momenteel al op een eenvoudige manier tekstbestanden inleest, dan kan dit wel eens de beste oplossing zijn bij koppelingen. Ga in dat geval niet aan beide systemen sleutelen om per se met bijvoorbeeld webservices te kunnen werken.
- Gebruik XML op het moment dat dit mogelijk is. XML is toekomstvast en flexibel. Let wel op de vorige opmerking: als het ontvangende systeem beter is in het inlezen van bijvoorbeeld csv-bestanden kan het beter zijn deze te gebruiken. In de volgende paragraaf meer over deze twee nieuwste ontwikkelingen die sterk gerelateerd zijn aan integratie.

De rol van XML en webservices

XML is al enkele jaren ‘hot’, om over webservices maar niet te spreken. Als momenteel een koppeling moet worden gebouwd, en performance staat het toe om gebruik te maken van XML, doe dit dan. Besef wel dat, hoewel XML bedoeld is als integratiemiddel, dit niet voor elke

koppeling is te prefereren. XML is namelijk tientallen keren langzamer is dan binair. Webservices zijn gedefinieerd om applicaties over Internetachtige omgevingen te koppelen. Ook hier geldt, als de omgeving en performance het toestaan, gebruik ze dan in verband met de toekomstvastheid (gebruik van wereldwijd geaccepteerde standaarden).
De theorie is mooi, maar hoe kan er in de praktijk, met behulp van Microsoft .NET geïntegreerd worden. Hieronder wordt een aantal korte voorbeelden gegeven van hoe er met Microsoft .NET op een eenvoudige wijze een koppeling gemaakt kan worden.

Message Queueing

Message Queueing klinkt triviaal, maar is waarschijnlijk het krachtigste instrument om te gebruiken bij integratie. Het is asynchroon, op een makkelijke manier te implementeren en het is gratis (standaardonderdeel van Windows). Onderstaande code toont hoe een bericht in een Message Queue verstuurd kan worden[2].

```
public void Send(string message)
{
  MessageQueue msgQueue =
new MessageQueue(@".\Private\
MijnQueue");
    msgQueue.Send(message);
}
```

Bovenstaand stukje code gaat er vanuit dat de Queue al bestaat. Overigens, om de queue, en dus het zenden en ontvangen transactioneel te maken, kan de queue transactioneel aangemaakt worden. Er zijn hiervoor geen codewijzigingen nodig.

Remote procedure calls

Door het aanroepen van een component van de andere applicatie kan ook tot integratie gekomen worden. Dit aanroepen kan op vele manieren, zoals het wrappen van COM-objecten, Remoting en het aanroepen van webservices. Het wrappen van COM-objecten is erg eenvoudig gemaakt, in feite is alleen een referentie naar het COM-object nodig. Vervolgens kan ermee gewerkt worden. Het aanroepen van objecten die op een remote server draaien, kan binnen Microsoft .NET op een tweetal manieren: door middel van remoting of door middel van webservices.

Webservices

Een andere veelgebruikte manier zal zijn het aanroepen van een webservice. Hieronder wordt in een kort stukje code een aanroep getoond van de Weather webservice die op Internet aanwezig is[3]. Na het aanmaken van een project, moet als eerste een webreferentie worden aangemaakt. Dit kan door in Visual Studio .NET in het Tools-menu voor de optie 'Add Web Reference' te kiezen.
Door in de dialoog de URL naar het asmx-bestand (http://www.learnxmlws.com/services/weatherRetriever.asmx) in te geven kan een referentie gemaakt worden naar de webservice. Vervolgens kan deze webservice in de code gebruikt worden.

Afbeelding 1. Toevoegen Web Referentie

```
public void CallWS()
{
  com.learnxmlws.www.Weather_
Retriever wr =
new com.learnxmlws.www.Weather_
Retriever();

  com.learnxmlws.www.Current_
Weather cw =
    wr.GetWeather("90210");
    MessageBox.Show(cw.Current_
Temp.ToString());
}
```

In feite wordt door de actie van het toevoegen van de webreferentie een proxy-class gegenereerd, die de daadwerkelijke aanroep doet naar de webservice. De code van deze proxy-class is te bekijken door in de Solution Explorer te kiezen voor 'Show all Files', vervolgens onder de webreferentie en vervolgens onder reference.map het reference.cs-bestand te openen.
Op exact eenzelfde manier is een koppeling te maken met de airport weather service die aangeboden wordt door http://www.capeclear.com/. De hier aangeboden webservice wordt aangeboden met een pakket dat CapeConnect heet, en is puur bedoeld voor het maken van webservices. Maak wederom een webreferentie aan, dit keer naar het WSDL-bestand op locatie http://live.capescience.com/wsdl/AirportWeather.wsdl.
Onderstaand stukje code toont een Message Box met hierin het weer op Schiphol, Amsterdam.

```
public void WeerSchiphol()
{
 com.capescience.live.Airport_
Weather aw =
new com.capescience.live._
AirportWeather();

com.capescience.live.Weather_
Summary ws =
   aw.getSummary("EHAM");

MessageBox.Show(ws.sky + " met " +
  ws.temp.ToString() + "
graden.");
}
```

Zoals in de hierboven twee genoemde voorbeelden is te zien, is de integratie van een webservice kinderspel. De echte uitdaging ligt in het aanbieden van diensten waarbij meer gedaan moet worden aan authenticatie en autorisatie. Dat dit al wel begint te ontstaan blijkt in de praktijk. Inmiddels bieden google[4] en Amazon[5] webservices aan die hun diensten ontsluiten.

Java webservices

Bij het ontsluiten van webservices die worden aangeboden vanuit een J2EE-

omgeving wordt in principe op dezelfde manier gewerkt. De problemen die hier bij komen kijken liggen vooral op het vlak van afstemming in de WSDL-files en de datatypes. Als eerste de WSDL-file. Als voorbeeld gebruiken we hier wederom de Weather-service, alleen dit keer geïmplementeerd in Java en aangeboden via Apache. Bij het analyseren van de WSDL-file[6] bij het toevoegen van de webreferentie worden geen methodes gevonden. Er wordt echter wel degelijk een webservice aangeboden. Om deze webservice te benaderen moeten we zelf een SOAP-proxy implementeren. Om dit te doen, maken we gebruik van de SOAP-functionaliteit in het Microsoft .NET Framework.

```
- <message name="getTemp
Request">
  <part name="zipcode"
type="xsd:string" />
  </message>
- <message name="getTemp
Response">
  <part name="return"
type="xsd:float" />
  </message>
- <portType name="Temperature
PortType">
- <operation name="getTemp">
  <input message="tns:getTemp
Request" />
  <output message="tns:getTemp
Response" />
  </operation>
  </portType>
```

In het WSDL-bestand dat we hebben opgehaald, vinden we een methode met de naam 'getTemp' (te vinden door te zoeken naar de XML-node '<portType>', en vervolgens hierbij de child-node '<operation name>'. Hieronder worden de input-message en de output-message aangegeven, die vlak hiervoor gedefinieerd zijn. Dit leert ons dat er een 'zipcode' in het formaat 'string' als invoerparameter geldt, en dat er een 'float' geretourneerd wordt.

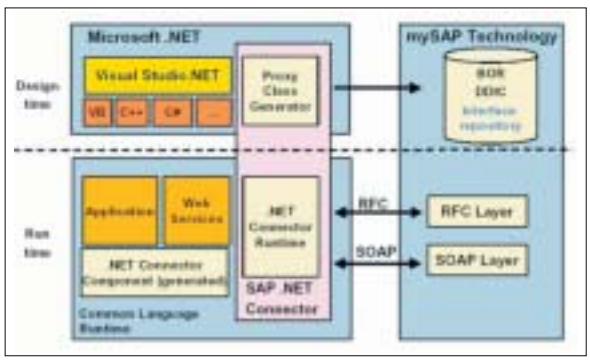
Nu is de geëigende manier van het aanmaken van een dergelijke proxy-class om dit door de wizard te laten doen, dat gaat in dit geval niet, aangezien er verscheidene WSDL-implementaties worden gebruikt. Nu kan de hier genoemde webservice wel aangeroepen worden. In dat geval zal evenwel een eigen proxy gebouwd moeten worden, waarbij het zelf opgestelde SOAP-bericht 'gepost' moet worden naar de webservice die hier wordt aangeboden.

Complex types

Naast de problemen met de WSDL-definitie-files, moet opgepast worden met complex types. Dit zijn types die samengesteld zijn uit andere types (denk aan structs). Het overkomen van deze complexe types gaat vaak niet goed, omdat de ontvangende kant deze niet begrijpt, of de volgorde verkeerd interpreteert. Om dit te voorkomen kan door middel van de standaardtypen worden gecommuniceerd. Dit is een veel veiligere optie.

Webservice-integratie

Het ontsluiten van andere systemen aan de hand van webservices is dus een reële optie. Hierbij zijn verschillen in implementatie van de WSDL-bestanden het grootste probleem[7]. Desondanks zullen deze de koppeling van de toekomst zijn. Grotere leveranciers, zoals het verderop genoemde SAP, bieden inmiddels ook interfaces aan door middel van webservices; anderen zullen spoedig volgen. De momenteel voorkomende problemen in de webservice-integratie worden in de toekomst ongetwijfeld opgelost. Bij het bouwen van een systeem dat potentieel ontsloten moet worden, is rekening houden met webservices een waardevolle investering.



Afbeelding 2. Rechtstreekse koppeling met SAP vanuit .NET

.NET koppelen aan SAP

Om een koppeling te kunnen maken met SAP moet een programmeur een ingang vinden die voldoet aan de in SAP gestelde API's. Kon dit vroeger al door middel van COM-aanroepen en RFC-calls (rechtstreeks op de SAP API), dit kan tegenwoordig vanuit .NET ook rechtstreeks.

Zoals afbeelding 2 aantoont, kan een koppeling tegenwoordig met proxy-classes gemaakt worden. Deze proxy-classes worden door SAP aangeroepen (door middel van een Proxy Generator die 100% integreert in de Server Explorer van Visual Studio .NET). Op het vlak van integratie via webservices gebeurt momenteel veel, maar onthoud dat met name RFC-aanroepen aanmerkelijk sneller zijn dan de webservice-aanroepen.

Ondersteuning

De ontsluiting van andere systemen vanuit .NET zal in de toekomst door steeds meer partijen ondersteund worden. Webservices zullen hierbij, gezien hun makkelijke implementatie en grote voordelen[8] steeds vaker worden gebruikt. De huidige problemen die ontstaan bij de koppeling van verschillende systemen zullen in de toekomst snel verholpen worden, want als productleverancier wil je niet op een eiland blijven zitten als de business draait om geïntegreerde systemen. Microsoft doet met .NET in ieder geval een hele grote stap in het ondersteunen van integratie.

Noten

1 Bijvoorbeeld door de gegevens periodiek over te hevelen naar een andere database.
2 Referentie nodig naar System.Messaging.
3 Er wordt gebruik gemaakt van de Weather webservice die aangeboden worden via http://www.LearnXmlWS.com/services/weatherRetriever.asmx.
4 http://www.google.com/apis/index.html
5 http://associates.amazon.com/exec/panama/associates/ntg/browse/-/1067662/002-1812343-2721668
6 http://www.xmethods.net/sd/2001/TemperatureService.wsdl
7 Overigens gaat het integreren van de EarthQuake-service, te vinden op http://webservices.tei.or.th/getQuakeData.cfc?wsdl, zonder meer goed.
8 Denk bijvoorbeeld aan crossplatform ondersteuning en mogelijkheid tot transporteren over Internet.