De motor van webservices loopt op SOAP. Het Simple Object Access Protocol (SOAP) maakt het mogelijk dat webservices via het internet benaderd kunnen worden. SOAP is als volgt gedefinieerd bij het W3C: "SOAP is a lightweight protocol for exchange of information in a decentralized, distributed environment." SOAP is een manier om informatie uit te wisselen via het internet. SOAP is tevens simpel omdat het een eenvoudig berichtformaat voorschrijft over hoe componenten met elkaar kunnen communiceren. Het berichtformaat SOAP voldoet aan de XML standaard.

achtergrond

De motor van webservices

SOAP headers als ontsnappingsclausule

Naast de originele SOAP specificatie, versie 0.9, werd in september 1999 versie 1.1 gepubliceerd. Deze laatste wordt nu meestal gebruikt. De XML protocol werkgroep bij het W3C werkt momenteel aan de versies 1.2 en 1.3. Recent is er een nieuwe groep opgericht, genaamd de 'Webservice Interoperability Organization', die ervoor gaat zorgen dat de verschillende SOAP-versies interoperabel blijven; een noodzaak voor het toekomstig succes van webservices. Een SOAP-bericht bestaat uit drie delen: de envelope, de header en de body. De envelope omvat het bericht en geeft meta informatie over de inhoud van het bericht en de methode van verwerking. De body faciliteert het daadwerkelijke remote procedure call (RPC) mechanisme. De body beschrijft welke functie van een webservice je gaat aanroepen en welke bijbehorende data heen en weer gestuurd gaat worden. Oftewel de class, haar methode en parameters. De SOAP header staat centraal in het hiernavolgende. Met behulp van voorbeelden in Visual Studio .Net zal een mogelijkheid van SOAP headers worden verduidelijkt.

SOAP HEADER Dit is de officiële W3C beschrijving van de SOAP header: "SOAP provides a flexible mechanism for extending a message in a decentralized and modular way without prior knowledge between the communicating parties. Typical examples of extensions that can be implemented as header entries are authentication, transaction management, payment et cetera." De header hoeft niet aanwezig te zijn in het SOAP bericht. Als het aanwezig is wordt het toegepast voor uitbreidingen op de standaard mogelijkheden van SOAP. De volgende uitbreidingen behoren tot de mogelijkheden:

1 *Authentisering*; door bijvoorbeeld de gebruikersnaam en wachtwoord mee te geven in de header kan de webservice zelf bepalen of gebruik is toegestaan.
2 *Transactie management*; headers vormen de ideale drager voor transactionele informatie. De SOAP aanroep van een webservices beschrijft de aanroep van één methode in een webservices. Als meerdere methoden gezamenlijk een transactie vormen dan kan de meta data die hiervoor nodig is om deze methoden te relateren in de header worden opgenomen zonder de methode zelf daarmee te belasten.
3 *Betaalmechanismen*; in de header meta-informatie meegeven van de partij die gebruik maakt van de services. Bijvoorbeeld door het unieke nummer van de klant, dat is vastgelegd bij het afsluiten van het samenwerkingscontract, kan de webservice een 'pay per view' business model realiseren. Voor het gemak ga ik hier voorbij aan de security-aspecten die hierbij komen kijken, maar het moge duidelijk zijn dat daar goed over na moet worden gedacht.
4 *Differentiatie*; een gebruiker van een webservice met bijvoorbeeld een Amerikaanse bankrekening, hoeft niet noodzakelijkerwijs hetzelfde te bepalen als iemand met een Belgische bankrekening. De header is het ideale medium om differentiatie toe te passen.
5 *Constructor*; je kan de header gebruiken om de webservice in een bepaalde initiële stand te zetten. Roundtrips naar webservices zijn kostbaar. Daarom is het onverstandig om vier eigenschappen van een object in te stellen voordat je een bepaalde functie aanroept, dit zou namelijk vijf roundtrips betekenen. Door de waarden van deze eigenschappen met de header mee te geven kun je in één roundtrip de webservice configureren.

De grenzen van de mogelijkheden van headers beperken zich tot de fantasie van diegenen die er gebruik van maken. Hieronder volgt een W3C voorbeeld van transactie gegevens in een header.

```
<SOAP-ENV:header>
   <t:Transaction
      xmlns:t="some-URI" SOAP-ENV:mustUnderstand="1">
          5
   </t:Transaction>
</SOAP-ENV:header>
```

Niet standaard Momenteel bestaan er nog weinig voorgedefinieerde headers. Een gestandaardiseerde header bestaat zelfs helemaal niet. Dit zal in de toekomst gaan veranderen. De nieuwe versies van SOAP hebben vaak betrekking op de toepassing van headers. Het voordeel van de standaardisatie van SOAP is dat de integratie van verschillende applicaties eenvoudiger wordt. Maar in sommige gevallen is een uitbreiding nodig. Omdat deze uitbreiding niet gestandaardiseerd is wordt de integratie van verschillende applicaties ingewikkelder. De letter 'S' van SOAP maakt hier zijn naam niet waar. BizTalk server 2000 is een goed voorbeeld van een applicatie die gebruik maakt van SOAP maar door het gebruik van proprietary headers niet zonder slag of stoot aansluit op andere applicaties. In veel gevallen zal de uitbreiding van headers noodzakelijk blijken, maar wees voorzichtig.

SOAP mustUnderstand attribuut De ontwerpers van SOAP voorzagen de behoefte om bepaalde header-gegevens verplicht dan wel optioneel te maken. In het bovenstaande voorbeeld van de transactionele header is te zien dat het transactie-element binnen de header het attribuut 'mustUnderstand' de waarde 1 krijgt. Dit betekent dat de webservice die wordt aangeroepen dit headerelement als verplicht moet beschouwen. De waarde van de 'mustUnderstand' kan de waarde 0 of 1 aannemen, waar 1 'true' betekent. De waarde 0 is equivalent aan het afwezig zijn van deze attribuut waarde.

Momenteel is er discussie in het SOAP-team over het feit dat mustUnderstand niet dwingend genoeg is. Dat webservice een header-element moeten kunnen begrijpen betekent nog niet dat dezer er iets mee moeten doen. Daarom is er nu discussie om de extra attribuut 'mustProcess' toe te voegen.

SOAP actor attribuut Ik heb in een eerder stadium geschreven dat SOAP simpel is, dat is niet helemaal waar. Toepassing van het SOAP actor-attribuut is complex. Met behulp van deze attribuut waarde is het mogelijk om de routing van SOAP gegevens te manipuleren. In het normale geval verzorgt SOAP een remote procedure call tussen een client en een webservice, de client roept een functie aan en geeft eventueel een paar headergegevens mee. Met behulp van de SOAP actor kun je aangeven dat deze header-gegevens door een andere webservice moeten worden afgehandeld dan de webservice die de functie-aanroep ontvangt. Met de SOAP actor geef je aan wie de ontvanger is van een bepaald headergegeven.

Microsoft Visual Studio .Net Webservices zijn belangrijke, misschien wel de belangrijkste onderdelen van de Microsoftstrategie met betrekking tot .Net. Zoals je tot nu toe componenten, lees dll's, maakt met Visual Basic of Visual C++ kun je nu webservices maken met het in januari 2002 gepubliceerde Visual Studio .Net. Het maakt nu niet meer uit of je een dll aanroept op je lokale machine of in Kyoto, in het laatste geval zal de aanroep van de functie waarschijnlijk alleen iets langer duren. In Visual Studio .Net kun je via 'add references' verwijzen naar een component of een webservice. Als je dat hebt gedaan kun je gebruik maken van de functies uit deze objecten. Je hoeft niet te weten wat de namen van de functies zijn. Dat weet Visual Studio .Net voor jou. Met behulp van een voorbeeld worden de mogelijkheden van headers in Visual Studio .Net onder de loep genomen. Het voorbeeld heeft betrekking op een populaire toepassing van headers: authentisering.

Overerven SoapHeader In de webservice voeg je een class toe die het gedrag overerft van de SoapHeader base class. De base class is onderdeel van de System.Web.Services.Protocols namespace. De base class is een lege header. Door de publieke variabelen gebruikersnaam en wachtwoord aan deze class toe te voegen bouw je de basis voor een minimaal authentisering mechanisme.

```
Imports System.Web.Services
Imports System.Web.Services.Protocols

Public Class oHeader
  Inherits SoapHeader

  Public Username As String
  Public Password As String
End Class
```

Toevoegen attribuut aan functie Uiteraard zul je zelf nog een functie moeten toevoegen die wat gaat doen met deze publieke variabelen. .Net maakt gebruik van een 'attribute based' programmeertechniek om extra mogelijkheden toe te voegen aan de programmeertaal en de .Net classes. In Visual Basic .Net gebruik je tags om een attribuut te maken. Om een class een webservice te laten zijn hoef je alleen het attribuut <WebService> aan de class toe te voegen. Om een functie binnen de class via het internet aanroepbaar te maken voeg je het attribuut <WebMethod> toe. Als je aan deze functie de overgeërfde header class wilt toevoegen gebruik je de <SoapHeader> attribuut. De compiler pakt de .Net Attributes op. De compiler zal een extern object aanroepen die extra code genereert gebaseerd op de inhoud van deze attributen. Visual Studio .Net genereert proxy (op de client)

en stub (op de server) code die de afhandeling van de remote procedure call via over HTTP faciliteert. Hier hoef je als .Net programmeur niet over na te denken. Maar als je wilt kan het wel, want de gegenereerde code is geen black box en kan handmatig nog worden aangepast.

In de functie kan worden gekeken naar de inhoud van de gebruikersnaam en wachtwoord. In onderstaand voorbeeld wordt op zeer eenvoudige wijze gecontroleerd of het wachtwoord correct is. In de praktijk zul je meestal een database aanroepen om de authentisering uit te voeren.

```
<WebService(Namespace:=”http://mydomain.com”, Descrip-
tion:=”Class beschrijving.”)> Public Class TestIt
   Inherits WebService
  Public oHeaderMemberVariable As oHeader
  <WebMethod(Description:=”Simpele authentisering”),
SoapHeader(“oHeaderMemberVariable”, Direction:=Soap-
HeaderDirection.InOut, Required := True)> Public Func-
tion TestMethod() As String
      If (oHeaderMemberVariable.Username = “admin”) And
(oHeaderMemberVariable.Password = “password”) Then
          oHeaderMemberVariable.Username = “Welcome
Administrator”
          oHeaderMemberVariable.DidUnderstand = True
          Return “getest”
       End if
   End Function
End Class
```

Als je het voorbeeld goed bekijkt zijn er een aantal zaken toegevoegd aan de SoapHeader, namelijk de ‘SoapHeaderDirection’ en ‘Required’. Met de ‘SoapHeaderDirection’ kun je de richting van de SOAP header aangeven. De aanroep van een webservice kun je opdelen in drie delen: aanroep (In), verwerking en teruggeven resultaat (Out). Er zijn drie richtingen te onderscheiden bij Soap headers: In, InOut en Out. Bij In geeft de client de header alleen mee bij de aanroep, bij Out ontvangt de client van de server een header bijvoorbeeld een unieke identifier. Bij InOut gaat de header heen en terug. De Required attribuut is opgenomen om aan te geven in hoeverre voor die functie een bepaalde header verplicht moet worden meegestuurd door de client.

Aanroepen vanuit client Om de webservice aan te kunnen roepen zal een referentie moeten worden gelegd naar de webservice via ‘add web references’ in Visual Studio .Net. In het onderstaande voorbeeld is de gerefereerde webservice ‘SRM’ genoemd.

```
Dim oTestWebservice As New SRM.TestIt()
Dim oHeader As New SRM.oHeader()
oHeader.Username = “admin”
oHeader.Password = “password”
oHeader.MustUnderstand = True
oTestWebservice.oHeaderValue = oHeader
oTestWebservice.TestMethod()
```

Er wordt een instantie van de webservice TestIt aangemaakt. De header wordt geïnstantieerd en doorgegeven aan de webservice via de oHeaderValue eigenschap. Vervolgens is het mogelijk om de functie van de webservice aan te roepen, waarbij het authentiseringmechanisme in werking treedt. Door aan de clientkant aan te geven dat de header begrepen moet worden door de webservice (‘MustUnderstand’ eigenschap) wordt gebruik gemaakt van de SOAP header specificaties inzake het ‘MustUnderstand’ attribuut. De webservice zal aan moeten geven middels de ‘DidUnderstand’ eigenschap dat de header is begrepen, anders volgt een foutmelding. In ons voorbeeld zal dus een foutmelding volgen bij een niet geslaagde authentisering. Het is ook mogelijk om zelf een foutmelding te genereren met de SoapHeaderException class.

SOAP XML Om een inzicht te krijgen in de informatie die tussen de client en de webservice heen en weer wordt gestuurd zijn hieronder de XML data opgenomen.

```
Request
<?xml version=”1.0” encoding=”utf-8”?>
<soap:Envelope
xmlns:soap=”http://schemas.xmlsoap.org/soap/envelope/”
xmlns:xsi=”http://www.w3.org/2001/XMLSchema-instance”
xmlns:xsd=”http://www.w3.org/2001/XMLSchema”>
<soap:header><oHeader soap:mustUnderstand=”1”
xmlns=”http://mydomain.com”>
<Username>admin</Username>
<Password>password</Password>
</oHeader></soap:header>
<soap:Body><TestMethod xmlns=”http://mydomain.com” />
</soap:Body>
</soap:Envelope>
Response
<?xml version=”1.0” encoding=”utf-8”?><soap:Envelope
xmlns:soap=”http://schemas.xmlsoap.org/soap/envelope/”
xmlns:xsi=”http://www.w3.org/2001/XMLSchema-instance”
xmlns:xsd=”http://www.w3.org/2001/XMLSchema”>
<soap:header><oHeader soap:mustUnderstand=”1”
xmlns=”http://mydomain.com”>
<Username>Welcome Administrator</Username>
<Password>password</Password></oHeader>
</soap:header>
<soap:Body>
<TestMethodResponse xmlns=”http://mydomain.com”>
<TestMethodResult>getest</TestMethodResult>
</TestMethodResponse>
</soap:Body>
</soap:Envelope>
```

Als programmeur hoef je geen seconde na te denken over de inhoud van deze XML berichten want die wordt door de .Net webclasses gegenereerd.

Conclusie

Headers zijn een belangrijk onderdeel van het SOAP protocol. Met headers kun je de functionaliteiten toevoegen aan het standaard remote procedure call mechanisme waar SOAP voornamelijk voor wordt gebruikt. Voorbeelden van handige toepassingen van headers zijn authentisering, transactie management of een constructormechanisme. In principe zijn de mogelijkheden van headers onbegrensd. Dit is een voordeel omdat het een ontsnappingsclausule is voor de beperkingen van het SOAP-protocol. Het is echter ook een nadeel omdat je met headers proprietary-oplossingen kan bouwen. Hopelijk leidt dit niet tot een wildgroei aan headertoepassingen, die allemaal weer op een andere manier geïmplementeerd zijn. Standaardisatie op het gebied van headers is wenselijk.

Edwin Jongsma

Jongsma is software architect en werkt bij Cap Gemini Ernst & Young.