SOAP staat voor Simple Object Access Protocol en beschrijft het XML-formaat van een bestand dat gebruikt wordt om een object aan te roepen. Met SOAP is het mogelijk om een request-replymechanisme te realiseren tussen een client en een server. Een client maakt een SOAP-bericht en stuurt dat naar een server object. De server stuurt een SOAP bericht terug om te vertellen wat er op de server heeft plaatsgevonden. De SOAP-berichtenstructuur schrijft niets voor over de implementatie van dit request-replymechanisme. Dit mechanisme is zelf geen onderdeel van de SOAP-standaard en er zijn meerdere implementaties van dit mechanisme mogelijk.

Hardcore SOAP: de Soap extensions

Aanvullende functionaliteit

In de meeste gevallen hoeft een software-ontwikkelaar geen verstand te hebben van de SOAP-berichtenstructuur of van het gebruikte request-reply mechanisme omdat de meeste ontwikkelomgevingen dit hebben ingebouwd. Maar het kan voordelig zijn om in te grijpen in de fasen van het request-replyproces. Hoe je dit kunt doen met behulp van SOAP Extensions staat centraal in dit artikel. Hier volgen een paar voorbeelden waar ingrijpen in het request-replyproces voordelig kan zijn:

Encryptie; er zijn kritische geluiden dat SOAP geen veilige standaard is. SOAP berichten worden via het internet verstuurd en zouden gemakkelijk kunnen worden onderschept. Deze kritiek is echter niet terecht want SOAP is zelf niet verantwoordelijk voor de veiligheid. SOAP-berichten kunnen door SOAP extensions worden beveiligd. Je kunt je voorstellen dat een productbestelling via internet, waar een creditcardnummer als parameter binnen SOAP wordt doorgegeven, door de client moet kunnen worden versleuteld. De server zal deze berichten vervolgens weer moeten kunnen ontsleutelen.

Compressie; SOAP-berichten voldoen aan de XML-standaard. XML heeft voordelen in verband met flexibiliteit, maar is niet altijd efficiënt als het gaat om bestandsgrootte. Dit komt doordat in XML een deel van de metadata van de gegevens worden meegestuurd. Door de SOAP-berichten op de client te comprimeren en op de server weer uit te pakken kan dit 'probleem' deels worden omzeild.

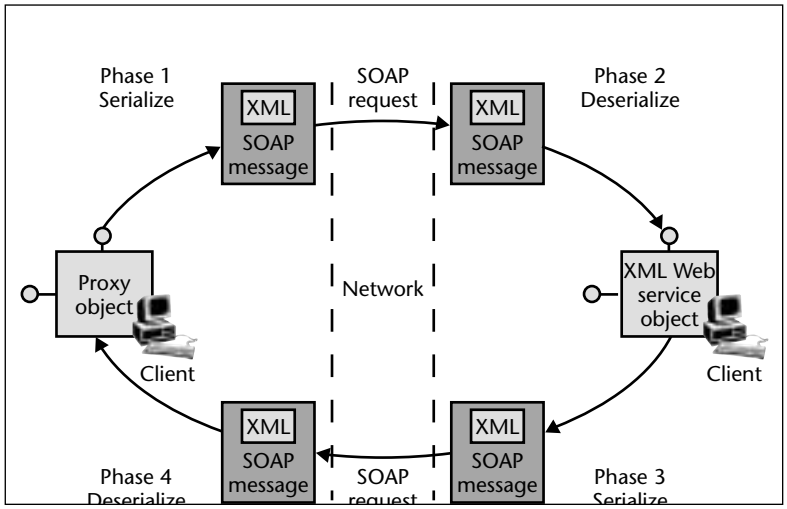
Logging; om meerdere redenen kan er behoefte zijn om de SOAP-berichten tijdens het request-replyproces te loggen.

Object state; objecten remote benaderen gebeurt meestal stateless, soms is het echter uit performance-perspectief wenselijk 'state' vast te houden op de server.

Belangrijke onderdelen van het request-replyproces zijn de transportlaag waarover SOAP-berichten worden verstuurd, de request-logica op de client en de reply-logica op de server. Hoe je in kunt grijpen in het request-replymechanisme met behulp van SOAP Extensions binnen Visual Studio .Net kunt u lezen in dit artikel.

> Er zijn kritische geluiden dat SOAP geen veilige standaard is

TRANSPORT In de praktijk maken de meeste SOAP-implementaties gebruik van het Hypertext Transfer



FIGUUR 1. Dit schema is afkomstig uit de Microsoft Developer Network (MSDN) helpfiles

Protocol (HTTP) voor het transport van berichten. Dit maakt namelijk request-reply via het internet mogelijk en dat is waar SOAP voornamelijk zijn toegevoegde waarde heeft ten opzichte van alternatieve request-replyoplossingen zoals DCOM en Corba. Behalve transport over HTTP is transport over HTTPS ook mogelijk.

Een ingreep in het serializationproces vereist een kopie van het SOAP-bericht

De HTTPS-transportlaag voegt SSL encryptie toe aan HTTP, dit maakt het mogelijk om SOAP-berichten veilig over het internet te versturen. Er zijn ook SOAP- implementaties die gebruik maken van message queuing en STMP voor transport. Deze implementaties zullen echter in het navolgende niet meer aan bod komen.

REQUEST REPLY MECHANISME Figuur 1 geeft een overzicht van de onderdelen betrokken bij het request-replymechanisme dat gebruik maakt van het HTTP protocol.

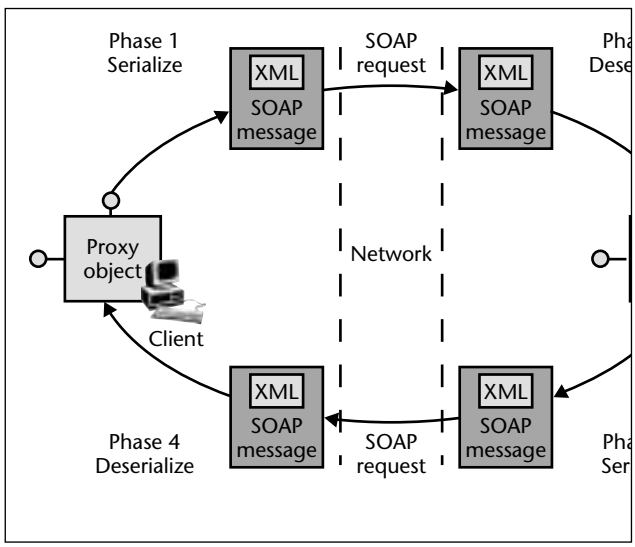
REQUEST - PROXY CLIENT Op de client draait een proxy object dat zich voordoet als het server object. Client-applicaties kunnen gebruik maken van het proxy object als draaide het remote object op de client. Het is deze proxyclient die ervoor zorgt dat er een SOAP-bericht wordt gestuurd over HTTP en die wacht op een reply van het server object.

REPLY - SERVER OBJECT (WEBSERVICE) De meeste SOAP-oplossingen zijn gebaseerd op transport over HTTP. Daarom zal dezelfde webserver die ook HTML-pagina's verstuurd gebruikt worden om SOAP- berichten in XML-formaat te ontvangen en te versturen. De webserver geeft het SOAP-bericht door aan het server object (webservice) die zijn business logica uitvoert en het resultaat weer teruggeeft aan de webserver. De webserver zal dit bericht weer terugsturen via HTTP naar de client.

SERIALIZATION Een belangrijke verantwoordelijkheid van de client en de server is ervoor zorgen dragen dat de business logica opgeslagen in de client- en serverobjecten worden omgezet naar het SOAP-berichtenformaat. Dit omzetproces wordt 'serialization' genoemd. Serialization wordt toegepast om de state van een object op te slaan opdat het object later of in het geval van SOAP in de vorm van een server object weer kan worden geactiveerd. Deserialization is het omgekeerde waar het bericht weer wordt omgezet in een object. Figuur 2 geeft een overzicht.

Het eerder genoemde nut van SOAP extensions, inzake encryptie, compressie en logging, grijpt in in het serializationproces. De serializer zal, behalve SOAP/-XML-berichten aan te maken op de client, deze ook nog versleutelen of comprimeren en aan de serverkant zal het omgekeerde plaatsvinden in de serializer.

SOAP EXTENSIONS In Visual Studio .Net is het eenvoudig om webservices te maken en aan te roepen via het internet. Als je meer wilt met webservices kun je gebruik maken van SOAP Extensions. Door de SOAP Extensions classes (onderdeel van de Web.Services.Protocols namespace) te overerven, heb je de volgende mogelijkheden:



FIGUUR 2. Serialization is een omzetproces waarbij de businesslogica wordt omgezet in het SOAP-berichtenformaat.

Webservices state bewaren met behulp van caching van webservices properties. Ingrijpen in het serialization proces. Nu je weet wat je met SOAP Extensions kan doen is het tijd om te kijken hoe dat precies werkt in Visual Studio .Net. In het navolgende werken we een voorbeeld uit van een SOAP Extension op de webserver, hetzelfde voorbeeld geldt ook voor de client.

Om je SOAP Extension op de server te maken doe je het volgende:

- Maak een webservice met Visual studio .Net en voeg een webmethod toe aan deze class.
- Voeg een class toe aan de webservices die de SoapExtension overerft.
- Initialiseer eventueel een aantal webservices eigenschappen die je wilt laten cachen. Hiervoor moet je de verschillende initializermethoden en de SoapExtensionAttribute class overerven.
- Voordat het serializationproces plaatsvindt moet het oorspronkelijke SOAP-bericht worden bewaard en een nieuw bericht moet worden aangemaakt waarmee verder wordt gewerkt. Door de ChainStream methode te overerven krijg je toegang tot het SOAP-bericht.
- Verander het serializationproces door de ProcessMessage methode te overerven.
- Koppel de SoapExtension class aan het attribuut van de gewenste webmethoden. De SoapExtension geldt niet voor de gehele webservice maar alleen voor de methoden die je zelf configureert.

Hier volgt de source code voor de overerving van beide Extension classes en de verwijzing naar de SoapExtension vanuit een specifiek WebMethod attribuut van een webservice. Hetzelfde attribuut kan ook worden toegevoegd aan de proxy client methode.

```
<WebService()> _

Public Class TestIt
Inherits WebService
<WebMethod(TestExtension()> _
Public Function TestMethod() As Address
End Function
End Class

Public Class TestExtension
Inherits SoapExtension
End Class

<AttributeUsage(AttributeTargets.Method)> _
Public Class TestExtensionAttribute
Inherits SoapExtensionAttribute
End Class
```

INITIALISATIE Er zijn twee initialisatiemethoden: GetInitializer en Initializer. De allereerste keer dat een webservice wordt aangeroepen gaat de GetInitializer af. Als de GetInitializer is gekoppeld aan een SoapExtensionAttribute class dan wordt de data die wordt geïnitialiseerd in het cache opgeslagen. Het maakt niet uit welk type data wordt opgeslagen. Deze opgeslagen data worden via de Initializer methode weer gekoppeld aan een webservice, ook als deze webservice een tweede keer wordt aangeroepen. De Initializer wordt elke keer aangeroepen als een gekoppelde WebMethod wordt benaderd.

> XML is weliswaar flexibel, maar niet altijd efficiënt wat betreft bestandsgrootte

Het voordeel van SoapExtensionAttribute caching is performance omdat de eigenschappen niet opnieuw hoeven te worden geïnitialiseerd. In een webfarm scenario is caching niet aan te bevelen. Om van SoapExtensions gebruik te maken hoef je geen gebruik te maken van SoapExtensionAttribute. Hier volgt een voorbeeld.

```
Private m_TestProperty As String
= “Cache deze datastring...”

Public Overrides ReadOnly Property Extension-
Type() As Type
Get
‘bepaal het datatype
Return GetType(TestExtension)
End Get
End Property

Public Overrides Property Priority() As
Integer
‘bepaal de prioriteit van de property
End Property

Public Property TestProperty() As String
Get
Return m_TestProperty
End Get
Set(ByVal Value As String)
m_TestProperty = Value
End Set
End Property
```

CHAINSTREAM Als je wilt ingrijpen in het serializationproces zul je een kopie moeten maken van het oor-

spronkelijke SOAP-bericht. Dit doe je door dit bericht te ketenen aan een lokale variabele in de ChainStream-methode. De returnwaarde van ChainStream is het SOAP-bericht dat je zelf kunt schrijven. Je gebruikt het originele bericht om de inhoud te lezen. Hier volgt een voorbeeld.

> In een webfarm scenario is caching niet aan te bevelen

```
Public Overrides Sub ProcessMessage(ByVal
message As SoapMessage)

Select Case message.Stage

Case SoapMessageStage.BeforeSerialize

Case SoapMessageStage.AfterSerialize

Case SoapMessageStage.BeforeDeserialize

Case SoapMessageStage.AfterDeserialize

Case Else
Throw New Exception("invalid stage")
End Select
End Sub
```

PROCESSMESSAGE Met behulp van ProcessMessage kun je ingrijpen in het serializationproces. Dit proces doorloopt vier stadia:

- voordat het object wordt gedeserialiseerd uit het request-SOAP-bericht;
- en erna;
- voordat het reply SOAP bericht is geserialiseerd;
- en erna.

Kijk voor de duidelijkheid nog een keer naar de illustraties. Als een SOAP-bericht na serialization op de client is gecomprimeerd zal het voor deserialization op de server weer moeten worden uitgepakt. De ProcesMessage zal hiervoor op de client en de server moeten worden overgeërfd. Hier volgt een voorbeeld.

```
Public Overrides Function ChainStream(ByVal
stream As Stream) As Stream
m_oldStream = stream
m_newStream = New MemoryStream()
Return m_newStream
End Function
```

CONCLUSIE In dit artikel staan SOAP Extensions centraal. De stappen die je moet nemen om in Visual Studio .Net je eigen Extensions te implementeren zijn te overzien, maar niet voor iedereen even gemakkelijk. Kennis van overerving en object oriëntatie zijn vereist. In de meeste gevallen zul je de mogelijkheden van SOAP Extensions niet nodig hebben. De behoefte aan compressie is meestal niet relevant omdat SOAP-berichten vaak niet groot zijn, meestal een stuk kleiner dan een HTML-pagina. Je eigen encryptie-extensie maken zou ik ook niet aanbevelen omdat het gemakkelijker is gebruik te maken van encryptie op de transportlaag via het HTTPS protocol. Logging en caching zouden wenselijk kunnen zijn. Er zijn nog veel meer mogelijkheden denkbaar. Het is belangrijk te weten dat je met SOAP Extensions in kunt grijpen in het serializationproces. Webservices worden steeds belangrijker, maar vroeg of laat lopen ontwikkelaars, die zich bezig houden met webservices, tegen beperkingen op en dan is het goed om te weten dat er SOAP Extensions zijn.

Edwin Jongsma
Jongsma is software architect en werkt bij Cap Gemini Ernst & Young. Dit artikel is het laatste uit een serie van drie over webservices.