

Välineet ja Web Services - WSDL-dokumentin generointi koodista ja päinvastoin

Versio 1.0

	SerAPI-projekti	
	Yhteyshenkilö	Heli Mäki (Heli.Maki@uku.fi)
	Dokumentin tila	Valmis
	Päiväys	27.5.2005

Sisällysluettelo

1	Johdanto	4
1.1	WSDL-dokumentin generointi.....	4
1.1.1	WSDL-dokumentin WS-I-yhteensopivuuden testaaminen.....	5
1.2	Web-sovelluspalvelun rungon luominen	5
1.3	Web-sovelluspalvelun käyttöönotto.....	5
2	Microsoft Visual Studio .NET 2003	7
2.1	WSDL-dokumentin generointi web-sovelluspalvelusta	7
2.2	Web-sovelluspalvelun toteuttaminen WSDL-dokumentin avulla	10
2.2.1	Tapa 1.....	11
2.2.2	Tapa 2.....	13
2.3	Web-sovelluspalvelun asiakassovelluksen toteuttaminen WSDL-dokumentin avulla	14
2.3.1	Tapa 1.....	15
2.3.2	Tapa 2.....	16
3	Oracle JDeveloper 10g.....	18
3.1	WSDL-dokumentin generointi web-sovelluspalvelusta	18
3.2	Web-sovelluspalvelun toteuttaminen WSDL-dokumentin avulla	19
3.3	Web-sovelluspalvelun asiakassovelluksen toteuttaminen WSDL-dokumentin avulla	21
4	Intersystems Caché 5.0	22
4.1	WSDL-dokumentin generointi web-sovelluspalvelusta	22
4.2	Web-sovelluspalvelun asiakassovelluksen toteuttaminen WSDL-dokumentin avulla	26
5	BEA WebLogic Workshop 8.1	27
5.1	WebLogic Workshop & web-sovelluspalvelujen toteuttaminen	27
5.2	WSDL-dokumentin generointi web-sovelluspalvelusta	27
5.3	Web-sovelluspalvelun toteuttaminen WSDL-dokumentin avulla	30
5.4	Web-sovelluspalvelun asiakassovelluksen toteuttaminen WSDL-dokumentin avulla	32
5.4.1	WebLogic Serverillä olevan web-sovelluspalvelun hyödyntäminen	32
5.4.2	Ulkopuolisen web-sovelluspalvelun hyödyntäminen WebLogic Workshopissa.....	34
5.5	BEA & standardointi.....	36

Versiohistoria

Versio:	Pvm:	Laatijat:	Selitys:
Versio 0	22.2.2005	Heli Mäki	Dokumentin luonti
Versio 0.1	25.5.2005	Heli Mäki	Dokumentin runko ja pohjaa lukuun Microsoft Visual Studio .NET 2003
Versio 0.2	17.3.2005	Heli Mäki	Luonnos johdantoon ja Microsoft Visual Studio .NET 2003 -lukuun.
Versio 0.3	1.4.2005	Saara Savolainen	Luonnos Intersystems Caché 5.0 -lukuun.
Versio 0.4	7.4.2005	Marko Sormunen	Luonnos Oracle JDeveloper 10g -lukuun.
Versio 0.5	18.4.2005	Heli Mäki	Lukua Microsoft Visual Studio .NET 2003 muokattu.
Versio 0.6	18.4.2005	Mika Tuomainen	Luonnos BEA WebLogic Workshop 8.1 -lukuun.
Versio 0.7	25.4.2005	Saara Savolainen	Lukua Intersystems Caché 5.0 muokattu.
Versio 0.8	25.4.2005	Heli Mäki	Esipuhetta, johdantoa ja lukua Microsoft Visual Studio .NET 2003 muokattu.
Versio 0.9	27.4.2005	Marko Sormunen	Lukua Oracle JDeveloper 10g muokattu.
Versio 0.10	4.5.2005	Heli Mäki	Dokumenttia muokattu kommenttien perusteella.
Versio 0.11	6.5.2005	Marko Sormunen	Lisätty JDeveloper-lukuun esimerkkejä
Versio 0.12	12.5.2005	Heli Mäki	Lisäyksiä ja muokkausta lukuun Microsoft Visual Studio .NET 2003.
Versio 1.0	27.5.2005	Mika Tuomainen Heli Mäki	Lukua BEA WebLogic Workshop 8.1 muokattu (Tuomainen) ja dokumenttia viimeistelty (Mäki). Dokumentin ensimmäinen valmis versio.

Esipuhe

Tämä työ liittyy SerAPI-hankkeeseen (Palveluarkkitehtuuri ja Web-sovelluspalvelut Terveydenhuollon Ohjelmistotuotannossa ja -integraatiossa), jossa tutkitaan ja kehitetään web-sovelluspalvelujen ja palvelupohjaisen arkkitehtuurin hyödyntämistä terveydenhuollon tietojärjestelmätarpeisiin ja sovellusintegraatioon ja uusiin sekä olemassa oleviin ohjelmistotuotteisiin. Hanketta rahoittavat Tekes (päätös nro 40437/04) sekä joukko yrityksiä ja sairaanhoitopiirejä.

Tämä dokumentti on tarkoitettu sovelluskehittäjille SerAPI-hankkeessa tuotettavien WSDL-dokumenttien hyödyntämisen tueksi.

1 Johdanto

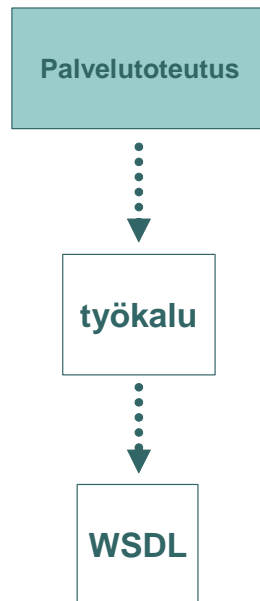
Tässä dokumentissa esitellään WSDL-dokumentin generointi web-sovelluspalvelusta sekä web-sovelluspalvelun ja web-sovelluspalvelun asiakasovelluksen toteuttaminen WSDL-dokumentin avulla käyttämällä eri sovelluskehittämiä, joita ovat:

- Microsoft Visual Studio .NET 2003
- Oracle JDeveloper 10g
- Intersystems Caché 5.0
- BEA WebLogic Workshop 8.1

Esimerkkinä käytetään EchoService-palvelua. Palvelu sisältää EchoText-metodin, jolle lähetetään merkkijono ja joka palauttaa merkkijonon. Metodin palauttama merkkijono sisältää merkkijonon "EchoText: " sekä syötteenä saadun merkkijonon, esim. lähetetään merkkijono "Hei" ja palvelu vastaa "Echoed text: Hei".

1.1 WSDL-dokumentin generointi

Web-sovelluspalvelua kuvaava WSDL-dokumentti generoidaan jostakin valmiista sovelluksesta tai sen komponentista (kuva 1). Generoitua WSDL-dokumenttia voidaan hyödyntää web-sovelluspalvelun toteutuksessa ja käyttöönotossa.



Kuva 1. WSDL-dokumentin generointi web-sovelluspalvelusta

1.1.1 WSDL-dokumentin WS-I-yhteensopivuuden testaaminen

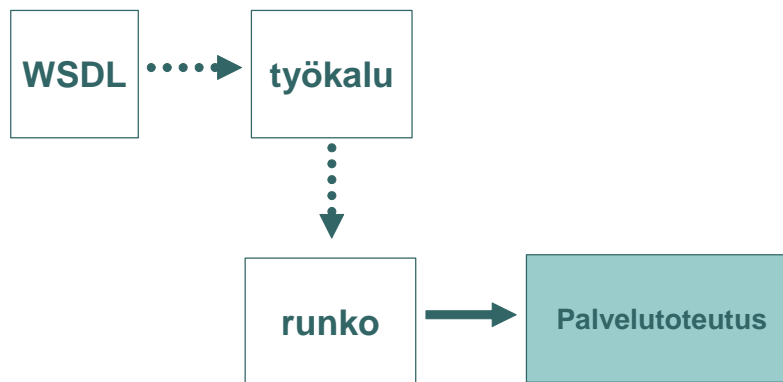
Web-sovelluspalvelusta generoidun WSDL-dokumentin WS-I-yhteensopivuus voidaan testata WS-I:n Interoperability Testing Tools 1.1 (SSBP) -työkalulla, josta on olemassa Java- ja C#-versiot:

- Java: http://www.ws-i.org/Testing/Tools/2004/12/SSBP_Java_Tools-BdAD.zip
- C#: http://www.ws-i.org/Testing/Tools/2004/12/SSBP_CS_Tools-BdAD.zip

WS-I-testaustyökalut on suunniteltu helpottamaan sovelluskehittäjiä määrittämään, onko heidän web-sovelluspalvelunsa WS-I-profiilin mukaisia. Interoperability Testing Tools 1.1 (SSBP) on suunniteltu Basic Profile 1.1:lle ja Simple SOAP Binding Profile 1.0:lle.

1.2 Web-sovelluspalvelun rungon luominen

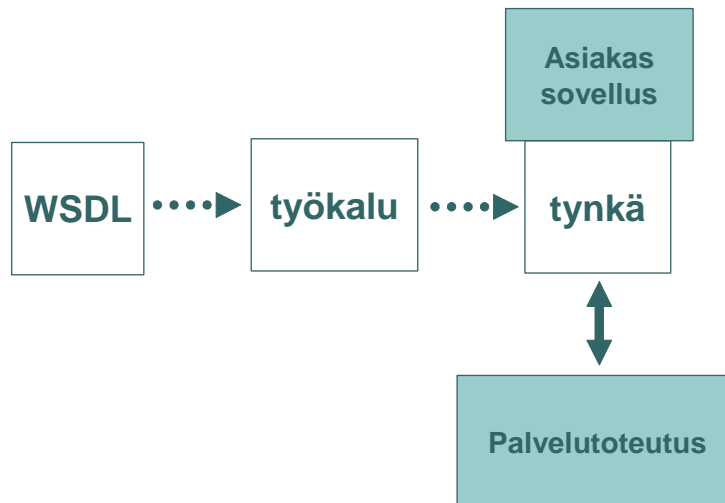
Web-sovelluspalvelun toteutukselle luodaan runko WSDL-dokumentista (kuva 2).



Kuva 2. Web-sovelluspalvelun rungon luominen WSDL-dokumentista

1.3 Web-sovelluspalvelun käyttöönotto

Ulkoinen web-sovelluspalvelu otetaan käyttöön WSDL-dokumentin avulla luomalla tynkä, jota asiakassovellus voi käyttää palvelun kutsumiseen (kuva 3).



Kuva 3. Asiakassovelluksen tynkän luominen WSDL-dokumentista

2 Microsoft Visual Studio .NET 2003

Tässä luvussa esitellään WSDL-dokumentin generointi web-sovelluspalvelusta sekä web-sovelluspalvelun ja web-sovelluspalvelun asiakassovelluksen toteuttaminen WSDL-dokumentin avulla Microsoft Visual Studio .NET 2003 -sovelluskehittäjä käyttämällä. Esimerkeissä käytetty ohjelmointikieli on C#.

Microsoft Visual Studio .NET 2003 Professional -sovelluskehittäjä ja C#-ohjelmointikieltä käytettäessä web-sovelluspalvelun projekti on tyypiltään Visual C# Project ja template ASP.NET Web Service. Web-sovelluspalvelun asiakassovelluksen projekti on myös tyypiltään Visual C# Project, mutta template on joko Windows Application tai ASP.NET Web Application.

2.1 WSDL-dokumentin generointi web-sovelluspalvelusta

Web-sovelluspalvelun dynaaminen WSDL generoituu automaattisesti, kun ASP.NET Web Service -tyyppinen projekti käännetään. Web-sovelluspalveluun tehtävät muutokset välittyvät myös dynaamiseen WSDL-dokumenttiin. Dynaamisen WSDL-dokumentin URL on web-sovelluspalvelun URL ja sen perässä "?wsdl" (esim. <http://localhost/EchoService/EchoService.asmx?wsdl>).

Web-sovelluspalvelusta voidaan luoda myös staattinen WSDL-dokumentti, johon web-sovelluspalveluun tehtävät muutokset eivät vaikuta. Staattinen WSDL luodaan tallentamalla dynaaminen WSDL paikalliselle levyille Web Services Discovery Tool (Disco.exe) -työkalulla antamalla komentokehoteessa (Visual Studio .NET 2003 Command Prompt) komento, joka on muodoltaan seuraavanlainen:

```
disco /out:directoryName URL
```

jossa

- /out: (lyhyt muoto /o:) on valinnainen optio, jolla voidaan määrittää kohdekansio ja *directoryName* on kohdekansion nimi. Oletusarvona on nykyinen kansio.
- URL on web-sovelluspalvelun (.asmx-tiedoston) URL

Lisätietoa Disco.exe:n muista parametreista löytyy Visual Studion dokumentaatiosta (Microsoft) osoitteesta:

```
ms-help://MS.VSCC.2003/MS.MSDNQTR.2003FEB.1033/cptools/html/cpgrfwebseVICESdiscoverytooldiscoexe.htm
```

tai komentokehoteessa komennolla:

```
disco /?
```

Disco.exe selvittää web-palvelimella (Web server) sijaitsevan web-sovelluspalvelun URL:n ja tallentaa web-sovelluspalveluun liittyvät dokumentit paikalliselle levyille. Disco.exe luo kohdekansioon .wsdl -tiedoston lisäksi tiedostot .disco ja results.discomap. Tiedosto .wsdl on WSDL-dokumentti, .disco discovery-dokumentti ja results.discomap XML-tiedosto, joka sisältää linkit web-sovelluspalvelun WSDL- ja discovery-dokumentteihin.

EchoService-palvelun toteuttavan .asmx.cs-tiedoston sisältö on esitelty esimerkissä 1.

Esimerkki 1. EchoService.asmx.cs-tiedoston sisältö

```
using System.ComponentModel;
using System.Web.Services;
using System.Xml.Serialization;

namespace EchoService
{
    [WebService(Namespace="urn:serapi:SOAPEXample")]
    public class EchoService : System.Web.Services.WebService
    {
        public EchoService()
        {
            InitializeComponent();
        }

        #region Component Designer generated code

        private IContainer components = null;

        private void InitializeComponent()
        {
        }

        protected override void Dispose( bool disposing )
        {
            if(disposing && components != null)
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #endregion

        [WebMethod]
        [return: XmlElement(ElementName="echoedText")]
        public string EchoText(string text)
        {
            return "Echoed text: " + text;
        }
    }
}
```

EchoService-palvelun kääntämisen jälkeen sen WSDL-dokumentti tallennetaan paikalliselle levyille siirtymällä komentokehotteessa kohdekansioon ja antamalla komento:

```
disco http://localhost/EchoService/EchoService.asmx
```

Komennon suorittamisen jälkeen sovellus palauttaa seuraavanlaisen ilmoituksen:

```
Disco found documents at the following URLs:
http://localhost/EchoService/EchoService.asmx?disco
http://localhost/EchoService/EchoService.asmx?wsdl
```

```
The following files hold the content found at the corresponding URLs:
.\EchoService.disco <- http://localhost/EchoService/EchoService.asmx?disco
.\EchoService.wsdl <- http://localhost/EchoService/EchoService.asmx?wsdl
The file .\results.discomap holds links to each of these files.
```

Esimerkissä 2 on EchoService-palvelun WSDL-dokumentin sisältö, esimerkissä 3 SOAP-palvelupyntö ja esimerkissä 4 SOAP-vastaus.

Esimerkki 2. EchoService.wsdl-tiedoston sisältö

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tns="urn:serapi:SOAPEXample"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
targetNamespace="urn:serapi:SOAPEXample"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:types>
    <s:schema elementFormDefault="qualified"
      targetNamespace="urn:serapi:SOAPEXample">
      <s:element name="EchoText">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="text"
              type="s:string" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="EchoTextResponse">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="echoedText"
              type="s:string" />
          </s:sequence>
        </s:complexType>
      </s:element>
    </s:schema>
  </wsdl:types>
  <wsdl:message name="EchoTextSoapIn">
    <wsdl:part name="parameters" element="tns:EchoText" />
  </wsdl:message>
  <wsdl:message name="EchoTextSoapOut">
    <wsdl:part name="parameters" element="tns:EchoTextResponse" />
  </wsdl:message>
  <wsdl:portType name="EchoServiceSoap">
    <wsdl:operation name="EchoText">
      <wsdl:input message="tns:EchoTextSoapIn" />
      <wsdl:output message="tns:EchoTextSoapOut" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="EchoServiceSoap" type="tns:EchoServiceSoap">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
      style="document" />
    <wsdl:operation name="EchoText">
      <soap:operation soapAction="urn:serapi:SOAPEXample/EchoText"
        style="document" />
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output>
```

```
        <soap:body use="literal" />
    </wsdl:output>
</wsdl:operation>
</wsdl:binding>
<wsdl:service name="EchoService">
    <documentation xmlns="http://schemas.xmlsoap.org/wsdl/" />
    <wsdl:port name="EchoServiceSoap" binding="tns:EchoServiceSoap">
        <soap:address location="http://localhost/EchoService/EchoService.asmx" />
    </wsdl:port>
</wsdl:service>
</wsdl:definitions>
```

Esimerkki 3. EchoService-palvelun SOAP-palvelupyyntö

```
POST /EchoService/EchoService.asmx HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "urn:serapi:SOAPEXample/EchoText"
```

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Body>
        <EchoText xmlns="urn:serapi:SOAPEXample">
            <text>string</text>
        </EchoText>
    </soap:Body>
</soap:Envelope>
```

Esimerkki 4. EchoService-palvelun SOAP-vastaus

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length
```

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Body>
        <EchoTextResponse xmlns="urn:serapi:SOAPEXample">
            <echoedText>string</echoedText>
        </EchoTextResponse>
    </soap:Body>
</soap:Envelope>
```

2.2 Web-sovelluspalvelun toteuttaminen WSDL-dokumentin avulla

Tässä luvussa esitellään web-sovelluspalvelun toteuttaminen Microsoft Visual Studio .NET 2003:lla kahdella eri tavalla. Tapa 2 on yksinkertaisempi, mutta toteuttamisessa käytettävän WSDL-dokumentin muuttuessa ohjelmakoodin päivittäminen on hankalampaa varsinkin laajempien web-sovelluspalveluiden tapauksessa.

2.2.1 Tapa 1

Web-sovelluspalvelu toteutetaan WSDL-dokumentin avulla seuraavasti:

1. Generoidaan WSDL-dokumentista web-sovelluspalvelun runkoluokka (Web Service stub file)
2. Käännetään runkoluokka DLL:ksi C# Compiler:lla (Csc.exe)
3. Luodaan Visual Studiolla uusi ASP.NET Web Service -projekti.
4. Lisätään projektiin viittaus (reference) luotuun DLL:ään.
5. Peritään .asmx.cs-tiedoston web-sovelluspalveluluokka runkoluokasta.
6. Lisätään web-sovelluspalveluluokan nimiavaruus
7. Ylikirjoitetaan .asmx-tiedoston web-sovelluspalveluluokassa runkoluokan web-metodit.
8. Lisätään web-metodeihin niiden toteutus.

Web-sovelluspalvelun runko (Web Service stub file) generoidaan WSDL-dokumentista Web Services Description Language Tool (Wsdlexe) -työkalulla antamalla komentokehoteessa (Visual Studio .NET 2003 Command Prompt) komento, joka on muodoltaan seuraavanlainen:

```
wsdl /server /language:language /out:filename /protocol:protocol  
/namespace:namespace path|URL
```

jossa

- **/server** on pakollinen optio, jolla määritetään, että generoidaan dokumentit web-sovelluspalvelua varten
- **/language:** (lyhyt muoto /l:) on valinnainen optio, jolla voidaan määrittää runkoluokan generoinnissa käytettävä kieli, ja *language* on käytettävä kieli. Oletusarvona on 'cs' (C#).
- **/out:** (lyhyt muoto /o:) on valinnainen optio, jolla voidaan määrittää generoitavan runkoluokan nimi, ja *filename* on runkoluokan (tiedoston) nimi. Oletusarvona on web-sovelluspalvelusta peritty nimi.
- **/protocol:** on valinnainen optio, jolla voidaan ylikirjoittaa käytettävä protokolla, ja *protocol* on protokolla.
- **/namespace:** (lyhyt muoto /n:) on valinnainen optio, jolla voidaan määrittää runkoluokan nimiavaruus, ja *namespace* on nimiavaruus. Oletusarvona on globaali nimiavaruus.
- *path* on staattisen WSDL-dokumentin sijainti.
- *URL* on dynaamisen WSDL-dokumentin URL.

Lisätietoa Wsdlexe:n muista parametreista löytyy Visual Studion dokumentaatiosta (Microsoft) osoitteesta:

```
ms-help://MS.VSCC.2003/MS.MSDNQTR.2003FEB.1033/cptools/html/  
cpgrfwebservicedescriptionlanguagetoolwsdlexe.htm
```

tai komentokehoteessa komennolla:

```
wsdl /?
```

EchoService-palvelun runkoluokka generoidaan staattisesta WSDL-dokumentista siirtymällä komentokehoteessa kohdekansioon ja antamalla komento:

```
wsdl /server /n:EchoServiceStub EchoService.wsdl
```

Seuraavaksi käännetään Wsdl.exe:n generoima runkoluokka DLL:ksi C# Compiler:lla (Csc.exe) antamalla komentokehoteessa komento, joka on muodoltaan seuraavanlainen:

```
csc /target:library /out:filename path
```

jossa

- **/target:** (lyhyt muoto /t:) on optio, jolla määritetään generoitavan tiedoston muoto, ja **library** määrittää luotavan tiedoston tyyppiä DLL.
- **/out:** on valinnainen optio, jolla määritetään generoitavan tiedoston nimi, ja *filename* on tiedoston nimi. Oletusarvona on runkoluokasta peritty nimi.
- *path* on runkoluokan (tiedoston) sijainti.

Lisätietoa Csc.exe:n muista parametreista löytyy Visual Studion dokumentaatiosta (Microsoft) osoitteesta:

```
ms-help://MS.VSCC.2003/MS.MSDNQTR.2003FEB.1033/cscomp/html/vcgrfBuildingFromCommandLine.htm
```

tai komentokehoteessa komennolla:

```
csc /?
```

EchoService-palvelun runkoluokka (EchoServiceStub.cs) käännetään DLL:ksi antamalla komentokehoteessa komento:

```
csc /t:library /out:EchoServiceStub.dll EchoService.cs
```

Luodaan uusi ASP.NET Web Service -projekti ja otetaan DLL käyttöön projektissa lisäämällä viittaus DLL:ään seuraavasti:

- Solution Explorer -ikkunassa klikataan projektin alla References-kohtaa hiiren oikealla ja valitaan "Add Reference...", jolloin Add Reference -ikkuna avautuu.
- Valitaan .NET-välilehti.
- Klikataan "Browse..."-painiketta ja etsitään sekä valitaan runkoluokasta käännetty DLL-tiedosto (esim. EchoServiceStub.dll), jolloin se tulee Selected Components -listaan.
- Klikataan OK-painiketta.

Sen jälkeen peritään .asmx.cs-tiedoston web-sovelluspalveluluokka DLL-tiedoston sisältämästä runkoluokasta, ylikirjoitetaan web-sovelluspalveluluokan web-metodit runkoluokan web-metodeista ja lisätään web-metodeihin niiden toteutus. Esimerkissä 5 on EchoService-palvelun toteuttavan .asmx.cs-tiedoston sisältö. Koodista on lihavoitu muokatut kohdat.

Esimerkki 5. EchoService.asmx.cs-tiedoston sisältö

```
using System.ComponentModel;
using System.Web.Services;

namespace EchoService
{
    [WebService(Namespace="urn:serapi:SOAPEXAMPLE")]
    public class EchoService : EchoServiceStub.EchoService
    {
        public EchoService()
    }
}
```

```
{
    InitializeComponent();
}

#region Component Designer generated code

private IContainer components = null;

private void InitializeComponent()
{
}

protected override void Dispose( bool disposing )
{
    if(disposing && components != null)
    {
        components.Dispose();
    }
    base.Dispose(disposing);
}

#endregion

[WebMethod]
public override string EchoText(string text)
{
    return "Echoed text: " + text;
}
}
```

EchoText-metodin vastaus palautuu oletuksena EchoTextResult-elementin sisältönä. Tämä elementti voidaan kuitenkin nimetä koodissa alkuperäisen WSDL-dokumentin mukaiseksi käyttämällä System.Xml.Serialization-nimiavaruuteen kuuluvaa [XmlElement]-attribuuttia (XmlElementAttribute-luokka):

```
[WebMethod]
[return: XmlElement(ElementName="echoedText")]
public override string EchoText(string text)
{
    return "Echoed text: " + text;
}
```

Myös toistuvien elementtien (maxOccurs="unbounded") tapauksessa elementit täytyy nimetä koodissa vastaamaan alkuperäistä WSDL-dokumenttia. Attribuutit voidaan nimetä vastaavasti [XmlAttribute]-attribuutin (XmlAttributeAttribute-luokka) avulla. Lisätietoa näistä sekä muista System.Xml.Serialization-nimiavaruuteen kuuluvista luokista löytyy Visual Studio dokumentaatiosta (Microsoft) osoitteesta:

```
ms-help://MS.VSCC.2003/MS.MSDNQTR.2003FEB.1033/cpref/html/
frlrfSystemXmlSerialization.htm
```

2.2.2 Tapa 2

Web-sovelluspalvelu voidaan toteuttaa WSDL-dokumentin avulla myös seuraavasti:

1. Generoidaan WSDL-dokumentista web-sovelluspalvelun runkoluokka (Web Service stub file), muuten kuten tavassa 1 (luku 2.2.1), mutta nimiavaruudeksi annetaan EchoService (wsdl /server /n:EchoService EchoService.wsdl)
2. Luodaan Visual Studiolla uusi ASP.NET Web Service -projekti.
3. Korvataan .asmx.cs-tiedoston sisältö runkoluokan sisällöllä.
4. Poistetaan web-sovelluspalveluluokan ja web-metodien abstract-määrittelyt.
5. Lisätään web-metodeihin niiden toteutus.

Esimerkissä 6 on esitelty EchoService.asmx.cs-tiedoston sisältö, joka on kopioitu runkoluokasta. Tämän jälkeen siihen on lisätty lihavoidut kohdat.

Esimerkki 6. Muokatun EchoService.asmx.cs-tiedoston sisältö

```
namespace EchoService
{
    using System.Diagnostics;
    using System.Xml.Serialization;
    using System;
    using System.Web.Services.Protocols;
    using System.ComponentModel;
    using System.Web.Services;

    /// <remarks/>
    [WebService(Namespace="urn:serapi:SOAPEXAMPLE")]
    [System.Web.Services.WebServiceBindingAttribute(Name="EchoServiceSoap",
Namespace="urn:serapi:SOAPEXAMPLE")]
    public class EchoService : System.Web.Services.WebService
    {
        /// <remarks/>
        [System.Web.Services.WebMethodAttribute()]
        [System.Web.Services.Protocols.SoapDocumentMethodAttribute("urn:serapi:SOAPEXAMPLE/EchoText", RequestNamespace="urn:serapi:SOAPEXAMPLE", ResponseNamespace="urn:serapi:SOAPEXAMPLE", Use=System.Web.Services.Description.SoapBindingUse.Literal, ParameterStyle=System.Web.Services.Protocols.SoapParameterStyle.Wrapped)]
        [return: System.Xml.Serialization.XmlElementAttribute("echoedText")]
        public string EchoText(string text)
        {
            return "Echoed text: " + text;
        }
    }
}
```

2.3 Web-sovelluspalvelun asiakassovelluksen toteuttaminen WSDL-dokumentin avulla

Tässä luvussa esitellään web-sovelluspalvelun asiakassovelluksen toteuttaminen Microsoft Visual Studio .NET 2003:lla kahdella eri tavalla. Tapa 2 on yksinkertaisempi, mutta sitä voidaan käyttää vain toteutettaessa asiakassovellusta dynaamisen WSDL-dokumentin avulla. Tapaa 1 voidaan käyttää asiakassovelluksen toteuttamisessa joko dynaamisen tai staattisen WSDL-dokumentin avulla.

2.3.1 Tapa 1

Web-sovelluspalvelun asiakassovellus toteutetaan WSDL-dokumentin avulla seuraavasti:

1. Generoidaan WSDL-dokumentista tynkäloukka (proxy class)
2. Käännetään tynkäloukka DLL:ksi C# Compiler:lla (Csc.exe)
3. Luodaan Visual Studiolla uusi Windows Application tai ASP.NET Web Application -projekti.
4. Lisätään projektiin viittaus (reference) DLL:ään.
5. Käytetään web-sovelluspalvelua DLL:n avulla.

Asiakassovelluksen tynkä (proxy class) generoidaan WSDL-dokumentista Web Services Description Language Tool (Wsd.exe) -työkalulla antamalla komentokehotteessa (Visual Studio .NET 2003 Command Prompt) komento, joka on muodoltaan seuraavanlainen:

```
wsdl /language:language /out:filename /protocol:protocol  
/namespace:namespace path|URL
```

jossa

- **/language:** (lyhyt muoto /l:) on valinnainen optio, jolla voidaan määrittää tynkäloukan generoinnissa käytettävä kieli, ja *language* on käytettävä kieli. Oletusarvona on 'cs' (C#).
- **/out:** (lyhyt muoto /o:) on valinnainen optio, jolla voidaan määrittää generoitavan tynkäloukan nimi, ja *filename* on proxy-loukan (tiedoston) nimi. Oletusarvona on web-sovelluspalvelusta peritty nimi.
- **/protocol:** on valinnainen optio, jolla voidaan ylikirjoittaa käytettävä protokolla, ja *protocol* on protokolla.
- **/namespace:** (lyhyt muoto /n:) on valinnainen optio, jolla voidaan määrittää tynkäloukan nimiavaruus, ja *namespace* on nimiavaruus. Oletusarvona on globaali nimiavaruus.
- *path* on staattisen WSDL-dokumentin sijainti.
- *URL* on dynaamisen WSDL-dokumentin URL.

Lisätietoa Wsd.exe:n muista parametreista löytyy mm. komennolla:

```
wsdl /?
```

EchoService-palvelun tynkäloukka generoidaan siirtymällä komentokehotteessa kohdekansioon ja antamalla komento:

```
wsdl /n:EchoService EchoService.wsdl
```

Seuraavaksi käännetään Wsd.exe:n generoima tynkäloukka DLL:ksi C# Compiler:lla (csc.exe) komennolla, joka on muodoltaan seuraavanlainen:

```
csc /target:option /out:filename path
```

jossa

- **/target:** (lyhyt muoto /t:) on optio, jolla määritetään generoitavan tiedoston muoto, ja *option* on tiedoston tyyppi. DLL:ää generoidessa tiedoston tyyppi on oltava *library*. Oletusarvona on 'exe'.
- **/out:** on valinnainen optio, jolla määritetään generoitavan tiedoston nimi, ja *filename* on tiedoston nimi. Oletusarvona on proxy-loukasta peritty nimi.

- `path` on proxy-luokan (tiedoston) sijainti.

Lisätietoa Csc.exe:n muista parametreista löytyy mm. komennolla:

```
csc /?
```

EchoService-palvelun tynkäloukka (EchoService.cs) käännetään DLL:ksi antamalla komentokehoteissa komento:

```
csc /t:library EchoService.cs
```

Luodaan uusi Windows Application tai ASP.NET Web Application -projekti ja lisätään projektiin viittaukset System.Web.Services.dll:ään ja tynkäloukasta käännettyyn DLL:ään seuraavasti:

- Solution Explorer -ikkunassa klikataan projektin alla References-kohtaa hiiren oikealla ja valitaan "Add Reference...", jolloin Add Reference -ikkuna avautuu.
- Valitaan .NET-välilehti.
- Valitaan listasta System.Web.Services.dll:ää ja klikataan Select-painiketta, jolloin DLL tulee Selected Components -listaan.
- Klikataan "Browse..."-painiketta ja etsitään sekä valitaan tynkäloukasta käännetty DLL-tiedosto (esim. EchoService.dll), jolloin se tulee Selected Components -listaan.
- Klikataan OK-painiketta.

Web-sovelluspalvelua käytetään DLL:n avulla. Esimerkiksi EchoService-palvelua käytetään seuraavasti:

```
EchoService.EchoService echoService = new EchoService.EchoService();  
this.lbOutput.Text = echoService.EchoText(this.tbInput.Text);
```

2.3.2 Tapa 2

Web-sovelluspalvelun asiakassovellus voidaan toteuttaa dynaamisen WSDL-dokumentin avulla myös seuraavasti:

1. Luodaan Visual Studiolla uusi Windows Application tai ASP.NET Web Application -projekti.
2. Lisätään projektiin viittaus (web reference) web-sovelluspalvelun dynaamiseen WSDL-dokumenttiin
3. Käytetään web-sovelluspalvelua viittauksen avulla.

Luodaan uusi Windows Application tai ASP.NET Web Application -projekti ja lisätään projektiin Web Reference web-sovelluspalveluun seuraavasti:

- Solution Explorer -ikkunassa klikataan projektin alla References-kohtaa hiiren oikealla ja valitaan "Add Web Reference...", jolloin Add Web Reference -ikkuna avautuu.
- Kirjoita URL-tekstikenttään web-sovelluspalvelun dynaamisen WSDL:n URL (esim. `http://localhost/EchoService/EchoService.asmx?wsdl`) ja klikkaa oikealla olevaa Go-painiketta.
- Kirjoita Web reference name: -tekstikenttään viittauksen nimi (esim. EchoService) ja klikkaa Add Reference -painiketta.

Web-sovelluspalvelua käytetään viittauksen avulla. Esimerkiksi EchoService-palvelua käytetään seuraavasti:

```
EchoService.EchoService echoService = new EchoService.EchoService();  
this.lbOutput.Text = echoService.EchoText(this.tbInput.Text);
```

3 Oracle JDeveloper 10g

Tässä kappaleessa käytetty JDeveloper-sovelluskehittimen versio on 10.1.2.0.0 (Build 1811). JDeveloper on varsin yleisesti käytetty Java-sovelluskehitin, joka tukee kaikkia yleisempiä Java-sovelluskehityksessä käytettyjä tekniikoita. JDeveloperin sovelluskehitystä helpottava toiminta perustuu ohjattujen työvaiheiden käyttöön uusia sovelluskohteita luotaessa. Käytännössä nämä työvaiheet luovat valmiiksi perusrakenteen laadittavalle kohteelle.

Lyhyesti lueteltuna JDeveloper tukee:

- Valmiin Java-luokan **julkistaminen** web-sovelluspalveluna, toisin sanoen luokan esittäminen WSDL-dokumentin avulla.
- Ulkopuolisia UDDI-rekistereitä web-sovelluspalveluja **käyttöönottaessa**.
- Java-luokkien **generointia** WSDL-dokumentista.
- Web-sovelluspalvelujen laatimista **visuaalisesti** UML-mallinnuksen avulla.

JDeveloperin toimintaa on esitelty laajasti Oraclen demonstraatioissa, jotka ovat saatavilla osoitteesta <http://www.oracle.com/technology/products/jdev/viewlets/viewlet.html>. Demonstraatiot sisältävät myös useita esimerkkejä web-sovelluspalveluiden laatimisesta. Saatavilla on myös perustietoa JDeveloperin käytöstä ja yleisestä Java-sovelluskehityksestä sen avulla.

3.1 WSDL-dokumentin generointi web-sovelluspalvelusta

WSDL-dokumentti voidaan generoida JDeveloperin ohjatun ”Create Java Web Service” -toiminnon avulla. Sen avulla mikä tahansa olemassa oleva julkinen Java-luokka voidaan muuntaa web-sovelluspalveluksi. JDeveloper laatii ohjatun toiminnon avulla web-sovelluspalvelulle WSDL-dokumentin ja kaikki tarvittavat tiedostot, joiden avulla web-sovelluspalvelu voidaan asentaa joko Oracle Application Server SOAP -palvelimeen tai Oracle J2EE Web Services (OC4J) -palvelimeen.

Web-sovelluspalvelun julkistaminen jaetaan kolmeen osioon:

1. Julkistettavan web-sovelluspalveluluokan valinta. Luokka voi olla mikä tahansa JavaBean-luokka, eli sen pitää vain olla julkinen.
2. Julkistettavien kutsujen valinta. Tässä yhteydessä voidaan myös valita, onko web-sovelluspalvelu tilallinen vai tilaton. Tilan säilyminen kutsujen välillä on toteutettu evästeiden avulla.
3. Web-sovelluspalvelun kutsupisteen määrittäminen, eli mistä URL:ista sovelluspalvelua voidaan kutsua.

Seuraavassa esimerkissä on esitetty JDeveloperin kehittämän EchoService.wsdl-tiedoston sisältö.

Esimerkki 7. EchoService.wsdl-tiedoston sisältö

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<!--Generated by the Oracle JDeveloper 10g Web Services WSDL Generator-->
<!--Date Created: Wed Apr 27 10:01:53 EEST 2005-->
<definitions
  name="MyWebService1"
```

```

targetNamespace="http://mypackage5/EchoServiceStub.wsdl"
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:tns="http://mypackage5/EchoServiceStub.wsdl"
xmlns:ns1="http://mypackage5/IMyWebService1.xsd">
<types>
  <schema
    targetNamespace="http://mypackage5/IMyWebService1.xsd"
    xmlns="http://www.w3.org/2001/XMLSchema"
    xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/" />
</types>
<message name="echoText0Request">
  <part name="text" type="xsd:string" />
</message>
<message name="echoText0Response">
  <part name="return" type="xsd:string" />
</message>
<portType name="EchoServiceStubPortType">
  <operation name="echoText">
    <input name="echoText0Request" message="tns:echoText0Request" />
    <output name="echoText0Response" message="tns:echoText0Response" />
  </operation>
</portType>
<binding name="EchoServiceStubBinding" type="tns:EchoServiceStubPortType">
  <soap:binding style="rpc" trans-
port="http://schemas.xmlsoap.org/soap/http" />
  <operation name="echoText">
    <soap:operation soapAction="" style="rpc" />
    <input name="echoText0Request">
      <soap:body use="encoded" namespace="MyWebService1" encodingSty-
le="http://schemas.xmlsoap.org/soap/encoding/" />
    </input>
    <output name="echoText0Response">
      <soap:body use="encoded" namespace="MyWebService1" encodingSty-
le="http://schemas.xmlsoap.org/soap/encoding/" />
    </output>
  </operation>
</binding>
<service name="MyWebService1">
  <documentation>Generated by the Oracle JDeveloper 10g Web Services
Stub/Skeleton Generator.
Date Created: Wed Apr 27 10:01:11 EEST 2005
WSDL URL: fi-
le:/C:/jdevl012/jdev/mywork/SerAPI/EchoServiceExample/classes/EchoService.wsdl
</documentation>
  <port name="EchoServiceStubPort" binding="tns:EchoServiceStubBinding">
    <soap:address location="http://laivuri27:8888/SerAPI-SOAP-testing-
context-root/MyWebService1" />
  </port>
</service>
</definitions>

```

3.2 Web-sovelluspalvelun toteuttaminen WSDL-dokumentin avulla

Web-sovelluspalvelun runko voidaan laatia JDeveloperin ohjatun ”Generate Web Service Stub/Skeleton” -toiminnon avulla. Oletuksena toiminto laatii tyngän, jonka avulla web-

sovelluspalvelua voidaan kutsua paikallisesta Java-sovelluksesta. Samaa toimintoa voidaan siis käyttää sekä web-sovelluspalvelun rungon että tyngän luontiin.

JDeveloperilla laadittu web-sovelluspalvelun runko sisältää ainoastaan interface-luokan, joka määrittelee web-sovelluspalvelun kutsurajapinnan. Tämän jälkeen tuota rajapintaa voidaan laajentaa web-sovelluspalvelun toteutukseksi ja se voidaan asentaa SOAP-palvelimeen kappaleessa 3.1 kuvulla tavalla.

Web-sovelluspalvelun rungon luominen jaetaan kahteen osioon:

1. Web-sovelluspalvelun WSDL:n hakeminen. WSDL voidaan joko hakea suoraan URL:in avulla tai käyttämällä ohjattua "Find Web Services" -toimintoa, jonka avulla WSDL voidaan hakea ulkopuolisesta UDDI-rekisteristä. Lisäksi ohjatulle toiminnolle pitää määrittellä, että halutaan nimenomaan web-sovelluspalvelun rungon laatiminen ("Generate Server-Side Skeletons" -toiminto).
2. Web-sovelluspalvelun runkoon laadittavien kutsujen valinta. Oletuksena kaikki WSDL-dokumentissa määritellyt kutsut ovat valittuina.

Esimerkki 8. EchoService-palvelun runko luotuna JDeveloperilla

```
package mypackage5;
import org.w3c.dom.Element;
/**
 * Generated by the Oracle JDeveloper 10g Web Services Stub/Skeleton Generator.
 * Date Created: Fri May 06 09:48:29 EEST 2005
 * WSDL URL:
file:/C:/jdev1012/jdev/mywork/SerAPI/EchoServiceExample/classes/EchoService.wsdl
 */

public interface EchoServiceSkeleton {
    public String echoText(String elem) throws Exception;
}
```

Rungosta saa laadittua toimivan web-sovelluspalvelun toteuttamalla JDeveloperin laatiman interface-luokan.

Esimerkki 9. Toteutettu EchoService-palvelu

```
package mypackage5;

public class EchoService implements EchoServiceSkeleton {
    public EchoService() {
    }

    public String echoText(String elem) throws Exception {
        return "Hello: "+elem;
    }
}
```

Valmiin palvelun saa toimimaan kappaleen 3.1 tavalla julkistamalla EchoService-palvelun toteutus web-sovelluspalveluna.

3.3 Web-sovelluspalvelun asiakassovelluksen toteuttaminen WSDL-dokumentin avulla

Web-sovelluspalvelun asiakassovellus(tynkä) voidaan laatia JDeveloperin ohjatun "Generate Web Service Stub/Skeleton" -toiminnon avulla. Samalla toiminnolla voidaan laatia myös web-sovelluspalvelun runko (kappale 3.2). Ohjatun toiminnon tuloksena syntynyttä luokkaa voidaan käyttää suoraan web-sovelluspalvelun kutsumiseen.

Web-sovelluspalvelun tyngän luominen jaetaan kahteen osioon:

1. Web-sovelluspalvelun WSDL:n hakeminen. WSDL voidaan joko hakea suoraan URL:n avulla tai käyttämällä ohjattua "Find Web Services" -toimintoa, jonka avulla WSDL voidaan hakea ulkopuolisesta UDDI-rekisteristä.
2. Web-sovelluspalvelun tynkään laadittavat kutsut. Oletuksena kaikki WSDL-dokumentissa määritellyt kutsut ovat valittuina.

4 Intersystems Caché 5.0

Tässä luvussa esitellään WSDL-dokumentin generointi web-sovelluspalvelusta sekä web-sovelluspalvelun toteuttaminen WSDL-dokumentin avulla. Sovelluskehittimenä oli Caché Studio ja esimerkissä käytetty ohjelmointikieli on Caché ObjectScript.

4.1 WSDL-dokumentin generointi web-sovelluspalvelusta

Caché web-sovelluspalvelutoteutuksen WSDL generoituu automaattisesti, kun web-sovelluspalveluksi määritelty Caché-luokka käännetään. WSDL-dokumentti on dynaaminen, eli web-sovelluspalveluun tehtävät muutokset välittyvät myös WSDL-dokumenttiin. WSDL:stä voidaan luoda myös staattinen XML-tiedosto käyttämällä FileWSDL-metodia (periytyy %SOAP.WebService-luokasta) esim.

```
Do ##class(EchoService.EchoService).FileWSDL("/EchoService.xml")
```

missä EchoService.EchoService on web-sovelluspalvelun luokka ja EchoService.xml on tiedosto, johon WSDL tallennetaan.

Kun Web-sovelluspalveluluokka käännetään, generoituu myös SOAP-rajapintaluokka kaikille palvelussa määritellyille metodeille. Tämä luokka vastaa palvelulle tulevien SOAP-pyyntöjen muuntamisesta web-metodikutsuiksi käyttämällä Cachén XML-to-object-muunnostekniikkaa.

Caché-luokka on määritelty web-sovelluspalveluksi, jos se periytyy %SOAP.WebService -luokasta (löytyy Caché-luokkakirjastosta). %SOAP.WebService-luokka sisältää kaiken toiminnallisuuden, jota tarvitaan palvelumetodien kutsumiseen SOAP-protokollan kautta. Web-sovelluspalvelun toteuttamat metodit määritellään ”web-metodeiksi” lisäämällä määrittelyjen loppuun ”WebMethod” -tunniste.

Uuden Web-sovelluspalvelun luominen tapahtuu Caché Studiolla seuraavasti:

- valitaan New-komento File-valikosta
- valitaan ”New Web Service”, jolloin saadaan näkyville ”Web Service Wizard”

Täytetään wizardiin luokan nimi ja seuraavat Web Service-parametrit:

- *Location* parametri määrittelee URL-osoitteen, josta asiakassovellus kutsuu palvelua. Tämä URL sisältyy web-sovelluspalvelun WSDL-dokumenttiin.
- *Namespace* on URI, jolla määritellään web-sovelluspalvelun nimialue. Oletuksena Web Service Wizard:issa tämä on ”http://tempuri.org”
- *Servicename* on palvelun nimi. Nimi on määriteltävä siten, että se alkaa kirjaimella ja sisältää vain aakkosnumeerisia merkkejä.

Alla on wizardilla luotu EchoService-palvelu, johon on lisätty web-metodin parametri ja metodin toteutus:

```
Class EchoService.EchoService Extends %SOAP.WebService [ProcedureBlock]
{
    Parameter SERVICENAME = "EchoService";
}
```

```
Parameter LOCATION = "http://localhost:1972/csp/echoservice";

Parameter NAMESPACE = "urn:serapi:SOAPEXample";

ClassMethod EchoText(text As %String) As %String [ WebMethod ]
{
    Quit text
}

}
```

EchoService-palvelun WSDL-dokumentin sisältö:

```
<?xml version='1.0' encoding='UTF-8' ?>
<definitions xmlns:http='http://schemas.xmlsoap.org/wsdl/http/'
xmlns:soap='http://schemas.xmlsoap.org/wsdl/soap/'
xmlns:s='http://www.w3.org/2001/XMLSchema'
xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
xmlns:s0='urn:serapi:SOAPEXample' xmlns:SOAP-
ENC='http://schemas.xmlsoap.org/soap/encoding/'
xmlns:mime='http://schemas.xmlsoap.org/wsdl/mime/' targetNamespace =
'urn:serapi:SOAPEXample' xmlns='http://schemas.xmlsoap.org/wsdl/'>
<types>
<s:schema elementFormDefault='qualified' targetNamespace =
'urn:serapi:SOAPEXample'>
<s:element name="EchoText">
<s:complexType>
<s:sequence>
<s:element name="text" type="s:string" minOccurs="0" />
</s:sequence>
</s:complexType>
</s:element>
<s:element name="EchoTextResponse">
<s:complexType>
<s:sequence>
<s:element name="EchoTextResult" type="s:string" minOccurs="0" />
</s:sequence>
</s:complexType>
</s:element>
</s:schema>
</types>
<message name="EchoTextSoapIn">
<part name="parameters" element="s0:EchoText" />
</message>
<message name="EchoTextSoapOut">
<part name="parameters" element="s0:EchoTextResponse" />
</message>
<portType name='EchoServiceSoap'>
<operation name='EchoText'>
<input message='s0:EchoTextSoapIn' />
<output message='s0:EchoTextSoapOut' />
</operation>
</portType>
<binding name='EchoServiceSoap' type='s0:EchoServiceSoap' >
<soap:binding transport='http://schemas.xmlsoap.org/soap/http'
style='document' />
<operation name='EchoText' >
<soap:operation
soapAction='urn:serapi:SOAPEXample/EchoService.EchoService.EchoText'
style='document' />
```

```
<input>
  <soap:body use='literal' />
</input>
<output>
  <soap:body use='literal' />
</output>
</operation>
</binding>
<service name='EchoService' >
  <port name='EchoServiceSoap' binding='s0:EchoServiceSoap' >
    <soap:address loca-
tion='http://localhost:1972/csp/echoservice/EchoService.EchoService.cls' />
  </port>
</service>
</definitions>
```

WSDL-dokumentin lisäksi web-sovelluspalvelulle generoituu automaattisesti luettelosivu (HTML), joka sisältää palvelun sekä sen metodien kuvaukset. Luettelosivu löytyy tämän esimerkin tapauksessa osoitteesta:

`http://localhost:1972/csp/EchoService/EchoService.EchoService.cls`

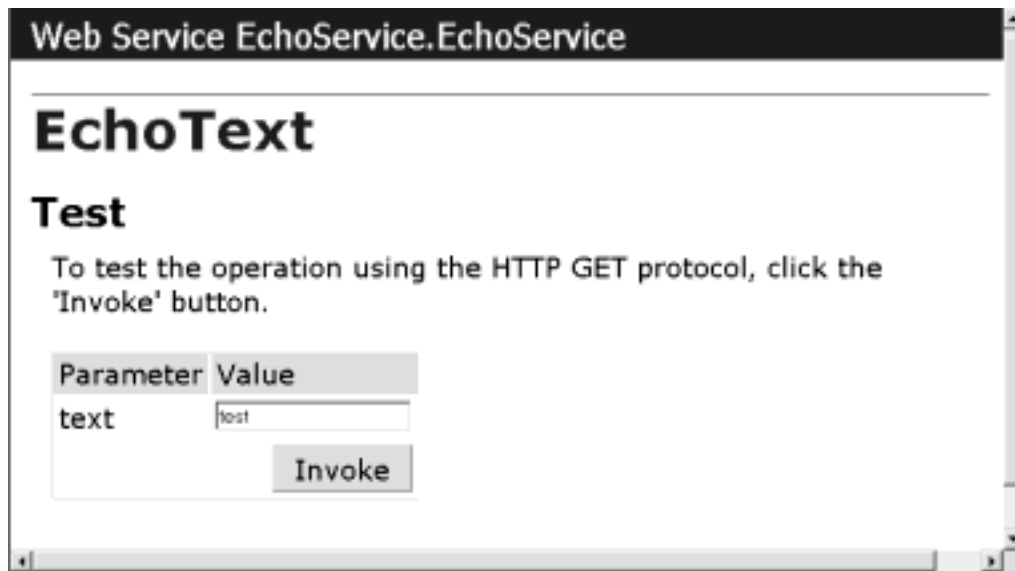
Tältä sivulta on myös linkki (ServiceDescription) WSDL-sivulle:

`http://localhost:1972/csp/EchoService/EchoService.EchoService.cls?WSDL`

sekä metodien testaussivulle, jolla on yksinkertainen käyttöliittymä metodien testaamista varten. Kuvissa 4-6 on esitetty EchoService-palvelun luettelo-sivu, EchoText-metodin testaussivu ja SOAP-vastaus metodikutsulle.



Kuva 4. EchoService-palvelun luettelosivu



Kuva 5. EchoService-palvelun EchoText-metodin testaussivu



Kuva 6. SOAP-vastaus EchoText-metodikutsulle

4.2 Web-sovelluspalvelun asiakassovelluksen toteuttaminen WSDL-dokumentin avulla

Caché-web-asiakassovelluksen on perittävä %SOAP.WebClient -luokka. SOAP-asiakasluokka sisältää palvelua vastaavat luokkametodit. Myös asiakassovelluksen metodit määritellään Web-metodeiksi lisäämällä ”WebMethod”-tunniste metodin määrittelyyn.

Asiakassovellus (SOAP-asiakas-luokka) voidaan toteuttaa WSDL-dokumentista käyttämällä SOAP-Client Wizardia. Tämä löytyy Cache Studio Tools valikosta Add Ins-toiminnosta. WSDL:n URL-osoitteen perusteella Wizard generoi asiakasluokat, jotka sisältävät palvelussa määriteltyjen web-metodien kutsut sekä mahdolliset muut web-sovelluspalvelun kutsumiseen tarvittavat luokat. EchoService-palvelun WSDL-dokumentista muodostettu asiakasluokka:

```
Class EchoService.EchoServiceSoap Extends %SOAP.WebClient
{
    Parameter LOCATION = "
http://localhost:1972/csp/echoservice/EchoService.EchoService.cls";

    Parameter NAMESPACE = " urn:serapi:SOAPEXample";

    Parameter SERVICENAME = "EchoService";

    Method EchoText(text As %String) As %String [ Final, ProcedureBlock = 1,
SoapBindingStyle = document, SoapBodyUse = literal, WebMethod ]
    {
        Quit ..WebMethod("EchoText").Invoke(##this,"http://
urn:serapi:SOAPEXample/EchoService.EchoService.EchoText",.text)
    }
}
```

Kun asiakasluokka käännetään, Caché-kääntäjä rakentaa automaattisesti SOAP-asiakasrajapinnan kaikille web-metodeille. Vastaavasti kuten SOAP-palvelurajapinta, SOAP-asiakasrajapinta on generoitu luokka, joka vastaa SOAP-pyyntöjen (http-XML-viestien) muodostamisesta käyttämällä Cachén XML-to-object -muunnostekniikkaa ja lähettämisestä web-sovelluspalvelulle.

5 BEA WebLogic Workshop 8.1

5.1 WebLogic Workshop & web-sovelluspalvelujen toteuttaminen

WebLogic Workshop Version 8.1.4 (WLW) on IDE, jolla toteutetaan J2EE sovelluksia WebLogic Platformille. WebLogic Workshopin avulla sovelluksia voidaan myös suunnitella, testata sekä jaella WebLogic Serverille.

WebLogic Workshop koostuu kahdesta keskeisestä teknologiasta: se on sekä integroitu sovelluskehitysympäristö että sovelluskehys, joka piilottaa kehittäjältä monet Java-kehitystyöhön liittyvät monimutkaiset asiat. IDE:ssä toteutettavat sovellukset voidaan rakentaa korkean tason komponenteista, näin välttää matalan tason API-kutsuilta. Tässä luvussa keskitytään korkeamman tason komponenttien hyödyntämiseen.

WLW:ssä toteutettavat web-sovelluspalvelut toteutetaan JWS-tiedostoon. JWS-tiedosto on syntaksiltaan puhdasta Javaa. Siinä kuitenkin käytetään attribuutteja, jotka hyödyntävät Workshopin ja WebLogic Serverin tarjoamia ominaisuuksia:

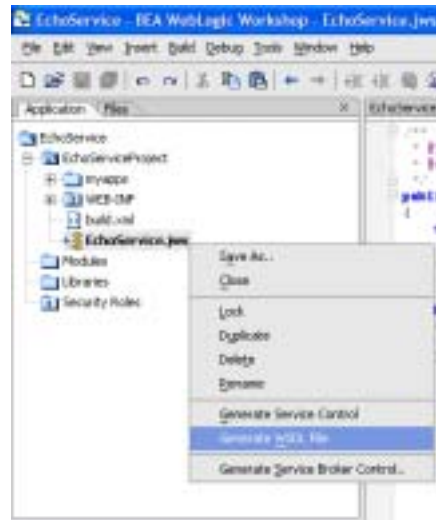
- JWS sisältää Javadoc-huomautuksia (annotations), joiden avulla päästään hyödyntämään WLW:n ajonaikaisen ympäristön web services -ominaisuuksia.
- JWS-tiedostopäätte kertoo WebLogic Serverille, että sovellusta pitää käsitellä web-sovelluspalveluna.

JWS-tiedosto sisältää ainoastaan web-sovelluspalvelun sovelluslogiikan, jota ollaan toteuttamassa. Web-sovelluspalvelun alla oleva infrastruktuuri, protokolla ja web-sovelluspalvelun elinkaaren hallinta käsitellään automaattisesti WebLogic Workshopin runtime:ssa.

Seuraavaksi käydään läpi, kuinka BEA WebLogic Workshop -sovelluskehittimellä toteutetaan luvussa 1 esitellyt web-sovelluspalveluiden toteutukseen ja hyödyntämiseen liittyvät vaiheet.

5.2 WSDL-dokumentin generointi web-sovelluspalvelusta

WSDL-dokumentti on helpointa generoida BEA WebLogic Workshopissa (WLW), kun web-sovelluspalvelun toteuttava sovellus on Application-ikkunassa auki (kuva 7).



Kuva 7. WSDL-dokumentin generointi JWS-tiedostosta

WSDL-dokumentin generoinnin vaiheet WLW:ssä:

1. Etsi Application-paneelistä EchoService.jws-tiedosto.
2. Klikkaa hiiren oikealla painikkeella tiedoston päällä ja valitse avautuvasta listasta "Generate WSDL File"
3. EchoService.jws-tiedostosta luodaan automaattisesti EchoServiceContract.wsdl-tiedosto samaan hakemistoon, jossa .jws-tiedosto sijaitsee.

WSDL nimetään automaattisesti seuraavasti:

PalvelunNimi.jws --> PalvelunNimi**Contract**.wsdl

EchoServiceContract.wsdl-tiedoston voi nimetä hakemistopuussa uudelleen (esim. yksinkertaisuuden vuoksi pelkästään EchoService.wsdl)

HUOM! Oletusarvoisesti JWS-tiedosto on linkitetty WSDL-tiedostoon. Näin JWS-tiedostoon tehdyt muutokset generoidaan automaattisesti myös WSDL-tiedostoon. Toisin päin muokkaaminen ei toimi, vaan käsin tehtävät muutokset WSDL-tiedostoon rikkovat linkin JWS-tiedostoon.

Workshopilla toteutetusta EchoService-palvelusta on alla generoitu WSDL-tiedosto. Web-sovelluspalveluun on konfiguroitu WLW:ssä asetukset, joissa sallitaan ainoastaan http-soap-protokolla ja target-namespace:ksi on asetettu urn:serapi:SOAPEExample.

```
<?xml version="1.0" encoding="utf-8"?>
<!-- @editor-info:link autogen="true" source="EchoService.jws" -->
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:conv="http://www.openuri.org/2002/04/soap/conversation/"
xmlns:cw="http://www.openuri.org/2002/04/wsdl/conversation/"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:jms="http://www.openuri.org/2002/04/wsdl/jms/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:s="http://www.w3.org/2001/XMLSchema" xmlns:s0="urn:serapi:SOAPEExample"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
```

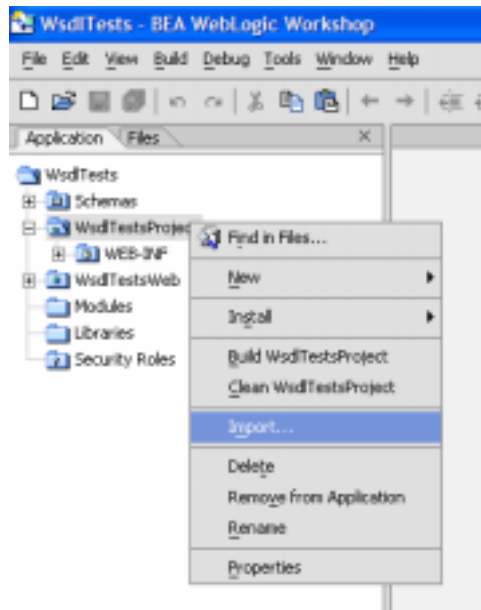
```
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" targetName-
space="urn:serapi:SOAPEXample">
  <types>
    <s:schema elementFormDefault="qualified" targetName
      space="urn:serapi:SOAPEXample" xmlns:s="http://www.w3.org/2001/XMLSchema">
      <s:element name="echoText">
        <s:complexType>
          <s:sequence>
            <s:element name="text" type="s:string" minOccurs="0"/>
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="echoTextResponse">
        <s:complexType>
          <s:sequence>
            <s:element name="echoTextResult" type="s:string" minOccurs="0"/>
          </s:sequence>
        </s:complexType>
      </s:element>
    </s:schema>
  </types>
  <message name="echoTextSoapIn">
    <part name="parameters" element="s0:echoText"/>
  </message>
  <message name="echoTextSoapOut">
    <part name="parameters" element="s0:echoTextResponse"/>
  </message>
  <portType name="EchoServiceSoap">
    <operation name="echoText">
      <input message="s0:echoTextSoapIn"/>
      <output message="s0:echoTextSoapOut"/>
    </operation>
  </portType>
  <binding name="EchoServiceSoap" type="s0:EchoServiceSoap">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
      style="document"/>
    <operation name="echoText">
      <soap:operation soapAction="urn:serapi:SOAPEXample/echoText"
        style="document"/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
  </binding>
  <service name="EchoService">
    <port name="EchoServiceSoap" binding="s0:EchoServiceSoap">
      <soap:address
        location="http://localhost:7001/EchoService/EchoService.jws"/>
    </port>
  </service>
</definitions>
```

WSDL-dokumentti on mahdollista generoida myös Ant-työkalulla komentorivillä (wsdlgen). Osoit-
teessa <http://e-docs.bea.com/wls/docs81/webserv/anttasks.html> on dokumentti "Web Service Ant
Tasks and Command-Line Utilities". Tässä dokumentissa ei kuitenkaan perehdytä kyseisen välineen
käyttöön.

5.3 Web-sovelluspalvelun toteuttaminen WSDL-dokumentin avulla

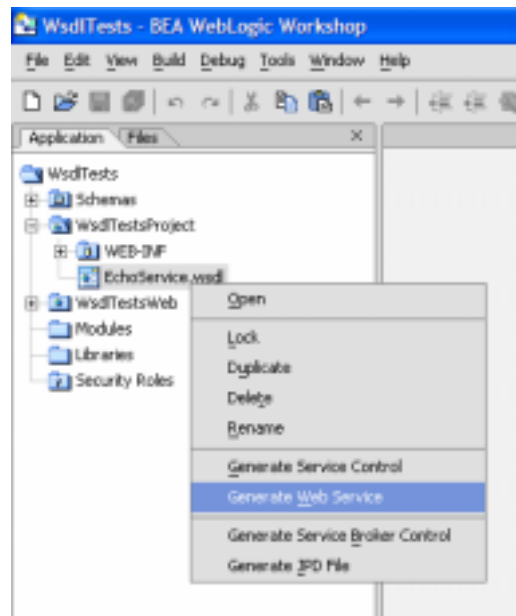
Web-sovelluspalvelun toteuttaminen WSDL-dokumentin pohjalta WLW:ssa:

1. Kopioidaan WSDL-dokumentti projektin hakemistoon joko resurssienhallinnan kautta tai WLW IDE:ssä import-komennolla (kuva 8)



Kuva 8. WSDL-dokumentin importtaus projektiin

2. Application-paneelissa paikallistetaan edellisessä vaiheessa projektiin liitetty WSDL-dokumentti.
3. Klikataan hiiren oikealla painikkeella WSDL-dokumenttia ja valitaan "Generate Web Service" (kuva 9).



Kuva 9. JWS-tiedoston generointi WSDL-tiedostosta

- Workshop generoi EchoService-nimisen web-sovelluspalvelun tiedostoon EchoService.jws

Alla on JWS-tiedostoon toteutettu palvelun runko:

```
public class EchoService implements com.bea.jws.WebService
{
    /**
     * @common:operation
     * @jws:protocol form-post="false" form-get="false"
     */
    public java.lang.String echoText (java.lang.String text)
    {
        // place your code here
    }

    static final long serialVersionUID = 1L;
}
```

Palvelun toteuttajan tarvitsee tässä yksinkertaisessa esimerkissä toteuttaa ainoastaan echoText-metodi. Kaikki muu web services -toiminnallisuus on piilotettu ohjelmoijalta.

Palvelun runko on mahdollista generoida WSDL-dokumentista myös Ant-työkalulla komentorivillä (wsdl2Service). Osoitteessa <http://e-docs.bea.com/wls/docs81/webserv/anttasks.html> on dokumentti "Web Service Ant Tasks and Command-Line Utilities". Tässä dokumentissa ei kuitenkaan perehdytä kyseisen välineen käyttöön.

5.4 Web-sovelluspalvelun asiakassovelluksen toteuttaminen WSDL-dokumentin avulla

Tässä luvussa käydään läpi, kuinka ulkopuolinen sovellus voi ottaa käyttöön WebLogic Serverillä olevan web-sovelluspalvelun käyttöön sekä kuinka WebLogic Workshopissa otetaan käyttöön ulkopuolinen web-sovelluspalvelu.

5.4.1 WebLogic Serverillä olevan web-sovelluspalvelun hyödyntäminen

WebLogic Serverillä oleva web-sovelluspalvelu tarjoaa sitä hyödyntäville sovelluksille web-sivun (kuva 10), joka löytyy esim. osoitteesta "http://localhost:7001/EchoService/EchoService.jws" eli "http://palvelunosoite:portti/palvelunnimi/palveluntoteuttavatiedosto".

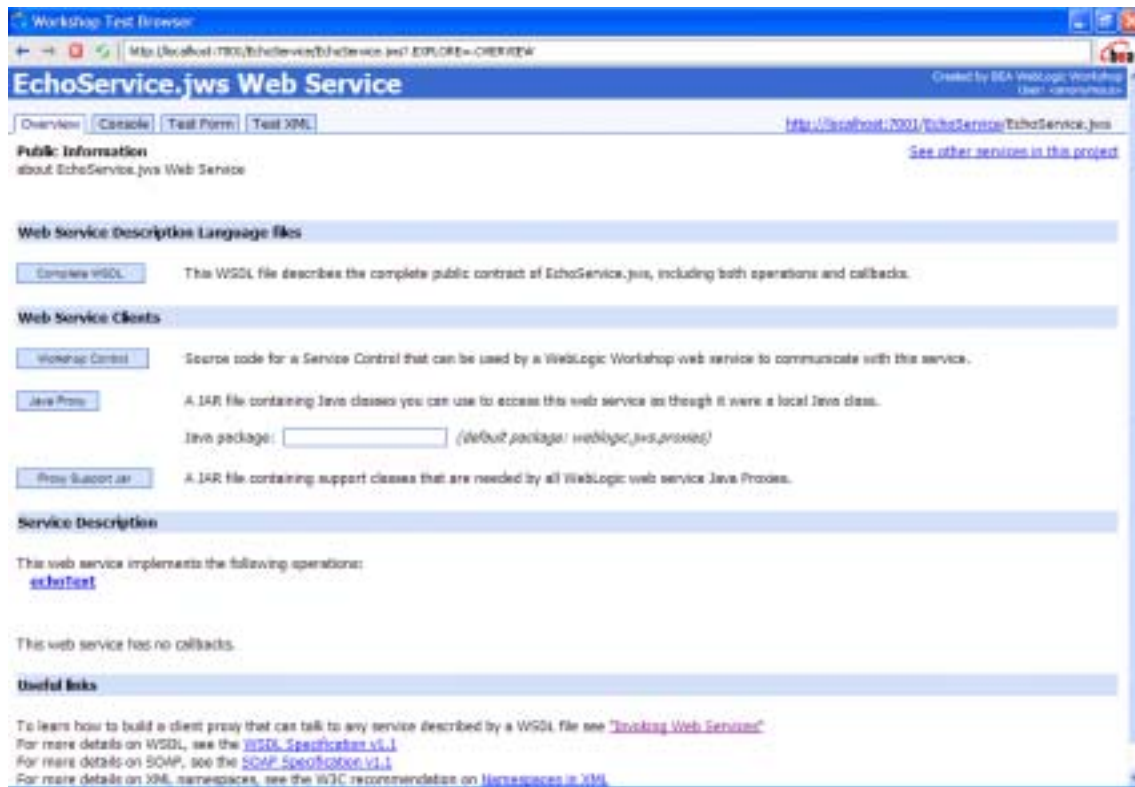
Palvelun web-sivulla on tarjolla WSDL-dokumentti sekä Java-asiakkaan toteuttajille suoraan tarjottavat tiedostot/paketit:

- Complete WSDL: Palvelun WSDL-dokumentti, jonka palvelun hyödyntäjä voi kopioida itselleen. Web-sovelluspalvelun WSDL-dokumentin saa myös lisäämällä palvelun osoitteen loppuun ?-merkin ja "WSDL"-tekstin:

`http://localhost:7001/EchoService/EchoService.jws?WSDL`

Esim. .NET-asiakkaan toteuttaja voi kopioida WSDL-dokumentin ja generoida siitä omilla apuvälineillään tynkälukan. Totta kai myös Java-asiakkaan toteuttaja voi toimia samalla tavoin.

- Workshop Control: Kontrolli, jota voidaan hyödyntää Workshopilla toteutettavassa sovelluksessa. Kontrollin avulla web-sovelluspalvelun käyttäminen on yksinkertaista. Web-sovelluspalveluiden hyödyntämisestä kontrollien avulla on kerrottu tarkemmin myöhemmin tässä luvussa.
- Java Proxy: JAR-paketti, joka sisältää Java-luokat, joiden avulla palvelua voidaan käyttää ikään kuin se olisi paikallinen Java-luokka. Käytännössä siis palvelusta generoitu tynkälukka.
- Proxy Support Jar: Tätä pakettia tarvitaan sovelluksissa, jotka käyttävät WebLogic:ssa kehitettyjen web-sovelluspalveluiden Java-proxyja. WebLogic servillä olevaa web-sovelluspalvelua hyödyntävän sovelluksen on siis käytettävä molempia luokkakirjastoja (tämä ja edellinen jar-paketti).



Kuva 10. Web-sovelluspalvelun sivusto

Edellisten lisäksi web-sovelluspalvelun sivustolta saadaan kuvaus web-sovelluspalvelun operaatioista (Service Description). Sivuston avulla voidaan myös testata web-sovelluspalvelua sekä tarkastella sille lähetettyjä ja sen palauttamia SOAP-viestejä.

Alla on esimerkki Java-asiakasohjelmasta (EchoClient.java), joka hyödyntää WebLogic Serverillä olevaa EchoService-palvelua.

```
// vaihe 1: tynkäluokka käyttöön
import weblogic.jws.proxies.*;

public class EchoClient
{
    public static void main(String[] args)
    {
        try
        {
            // vaihe 2: Luodaan proxy-luokasta olio.
            EchoService_Impl proxy = new EchoService_Impl();

            // vaihe 3: käytetään protokollana SOAP:ia
            // (BTW: muita tämä echoService ei tuekaan)
            EchoServiceSoap soapProxy = proxy.getEchoServiceSoap();

            // vaihe 4: Käytetään palvelun metodia echoText
            System.out.println(soapProxy.echoText("oukki doukki"));
        }
        catch (Exception e)
        {
```

```

        e.printStackTrace();
    }
}

```

Palvelun web-sivustolta saatavat Java Proxy- ja Proxy Support Jar -tiedostot on lisättävä luokkapolkuun (classpath) tai vastaavaan, josta web-sovelluspalvelua hyödyntävä sovellus ne löytää. Toistamalla edellisen koodissa olevat neljä vaihetta voidaan web-sovelluspalvelu ottaa käyttöön myös esim. JSP-sivuilla.

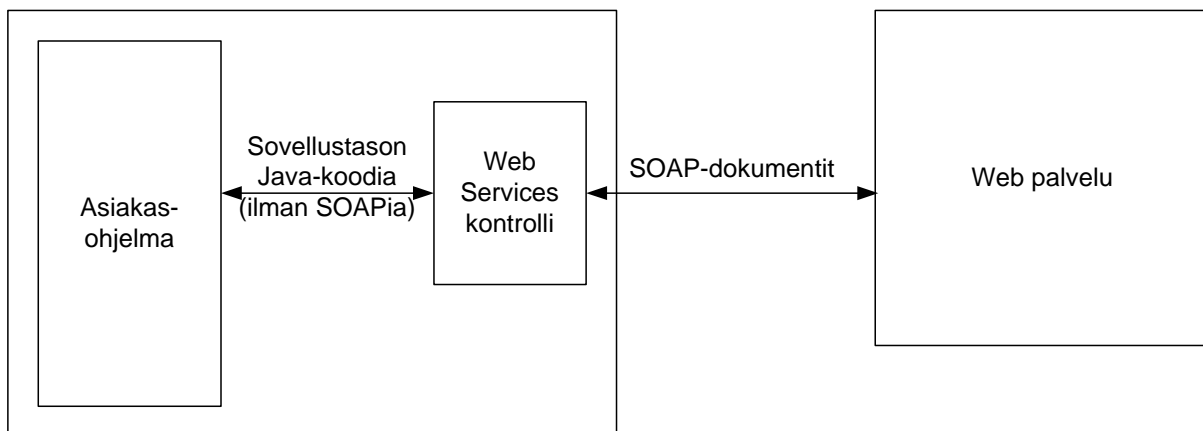
Java proxy -luokka on mahdollista generoida WSDL-dokumentista myös Ant-työkalulla komentorivillä (clientgen). Osoitteessa <http://e-docs.bea.com/wls/docs81/webserv/anttasks.html> on dokumentti "Web Service Ant Tasks and Command-Line Utilities". Tässä dokumentissa ei kuitenkaan perehdytä kyseiseen tapaan.

5.4.2 Ulkopuolisen web-sovelluspalvelun hyödyntäminen WebLogic Workshopissa

Web-sovelluspalveluiden hyödyntäminen Workshopissa kannattaa toteuttaa kontrollien avulla. Totta kai hyödyntäminen voidaan toteuttaa myös kopioimalla hyödynnettävän palvelun WSDL-tiedosto ja generoimalla siitä tynkäluokka Ant-työkalulla. Kontrollit tarjoavat kuitenkin huomattavasti yksinkertaisemman ja mielekkäämmän tavan kutsua web-sovelluspalveluita.

WebLogic Workshop tarjoaa valmiiksi sisäänrakennetun web services -kontrollin, jonka avulla WebLogic Workshop sovellusten on helppo päästä ulkopuolisiin web-sovelluspalveluihin. Web services -kontrolli voidaan luoda mistä tahansa web-sovelluspalvelusta, josta on julkaistu WSDL-dokumentti. Web services -kontrolli tarjoaa web-sovelluspalvelun ja palvelua hyödyntävän asiakkaan välille rajapinnan, jonka avulla asiakassovellus voi kutsua web-sovelluspalvelun metodeja ja käsitellä web-sovelluspalvelun vastauksia toteuttamalla ainoastaan sovellustason koodia. Asiakkaan ei tarvitse tietää viestien muodon eikä viestinnän protokollien yksityiskohtia.

Kuvassa 11 on kuvattu edellä kuvatun mukainen arkkitehtuuri.

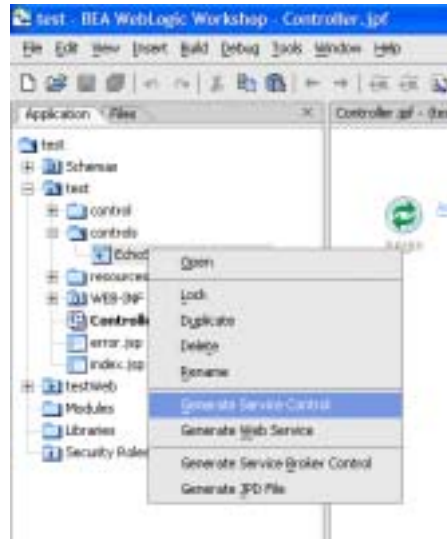


Kuva 11. Arkkitehtuuri, jossa hyödynnetään Web services -kontrollia

Web services -kontrolli käsittelee SOAP liikenteen web-sovelluspalvelun ja asiakassovelluksen välillä. Web services -kontrollissa voidaan hyödyntää myös toiminnallisuutta, jolla parsroidaan web-sovelluspalvelun palauttavat SOAP-dokumentit. Tämä toiminnallisuus voidaan toteuttaa WebLogic Workshopissa esim. XMLBeans:ien avulla.

Web services -kontrollin käyttäminen:

1. Ensin luodaan projektiin web services -kontrolli. Kontrollin voi luoda WebLogic Workshopissa kahdella tavalla:
 - a. Tuodaan WSDL-dokumentti sovellukseen joko lisäämällä se resurssienhallinnassa tai import-komennolla (kuten luvussa 5.3). Klikataan hiiren oikealla näppäimellä WSDL-tiedoston päällä ja valitaan valikosta komento "Generate Service Control" (kuva 12). WLW luo tällöin uuden web services -kontrollin.



Kuva 12. Web Services kontrollin luominen

- b. Valitaan Workshopin valikosta Insert-Controls-Web Service. Avautuvaan ikkunaan annetaan kontrollille nimi ja kerrotaan, mistä WSDL-dokumentti löytyy. Sitten valitaan Create ja WLW luo web services -kontrollin.
2. Liitetään edellä luotu kontrolli projektiin joko import-lauseella koodissa tai Workshopin Action View -näkyvässä raahaamalla kontrolli näkymäikkunaan.
3. EchoServices-palvelun kutsuminen Web services -kontrollin avulla
 - a. Luodaan kontrollimuuttuja

```
private control.EchoServiceControl echoServiceControl;
```
 - b. Kutsutaan web-sovelluspalvelua kontrollin kautta ja otetaan palvelun palauttama vastaus talteen vastaus-oliioon.

```
String vastaus = echoServiceControl.echoText("Haloo");
```

Tämän jälkeen vastausta voidaan hyödyntää koodissa halutulla tavalla.

Web services -kontrollia voi ja pitää konfiguroida sen mukaan, mitä protokollaa (soap, http-post, http-get, ..) ja mitä sanomamuotoa (document literal, SOAP RPC) web-sovelluspalvelu tukee. Nämä tiedot voi tarkastaa web-sovelluspalvelun WSDL-dokumentista. Workshop käyttää document literal -sanomamuotoa oletusarvoisesti ja tukee myös SOAP RPC:tä.

5.5 BEA & standardointi

BEA on mukana Apache Beehive -projektissa (<http://incubator.apache.org/beehive/>). Projektin tavoitteena on tarjota open source -pohjainen alusta J2EE-sovellusten ja palveluarkkitehtuurien rakentamiseen. Projektia on ilmoittanut tukevansa yli 50 ohjelmisto- ja alustatoimittajaa, muun muassa Borland, Capgemini, Compuware, Intel, mySQL, Red Hat, Salesforce.com ja VERITAS.

BEA on antanut osan Java-sovelluskehitysalustastaan Beehive open source -yhteisön käyttöön, muun muassa BEA:n Java annotations-, Java controls-, Java Web services- ja Java Page Flow -teknologiat. BEA WebLogic Workshopissa toteutetut sovellukset on näin mahdollista siirtää myös muihin ympäristöihin kuten Tomcat:lle. Tomcat vaatii luonnollisesti Beehive-asennuspaketin lisäämistä ajoympäristöön.

Beehive-projektin lisäksi BEA on liittynyt Eclipse -liittymään, jossa on tavoitteena luoda yhtenäinen avoimeen lähdekoodiin perustuva kehitysympäristö. BEA:n seuraavan sukupolven WebLogic Workshop -kehitysympäristö tulee hyödyntämään Eclipsen ominaisuuksia, laajennuksia ja sovel-luskehystä.

Lähteet

Microsoft Visual Studio .NET 2003 Documentation.

BEA. BEA WebLogic Workshop Help (Online). <http://e-docs.bea.com/workshop/docs81/doc/en/core/index.html>, 3.5.2005.