XSLT extensies in JDeveloper

De voor- en nadelen van XSLT

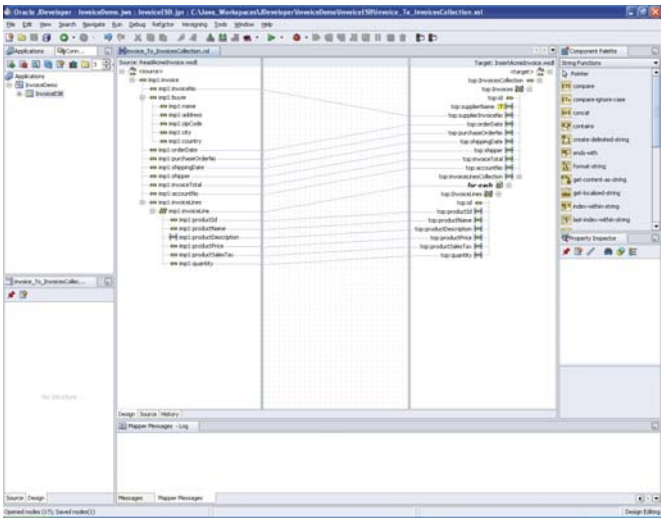
Dit artikel gaat over de mogelijkheid in Oracle JDeveloper en Oracle SOA Suite om methoden uit Java-klassen te gebruiken in XML-transformaties. In dit artikel wordt een eenvoudige Java-methode, die bankrekeningnummers in IBAN-formaat retourneert, als XSLT-extensie beschikbaar gesteld en gebruikt in een transformatie.

JDeveloper beschikt sinds een aantal versies over de XSL Map, een extensie waarmee transformaties tussen XML-documenten grafisch gedefinieerd worden. Deze transformaties worden als XSL-documenten opgeslagen.

De XSL Map ondersteunt naast de meeste standaard – door het World Wide Web Consortium (W3C) gedefinieerde – XSL(T) en XPath-constructies, ook Oracle-specifieke functies. Ook bestaat de mogelijkheid om methoden uit Java-klassen beschikbaar te stellen als XSLT-extensies. Deze kunnen design-time in de XSL Map gebruikt worden om transformaties te definiëren en run-time door Java-applicaties en ESB- en BPEL-projecten – draaiend in de SOA Suite – uitgevoerd worden in bestaande transformaties.

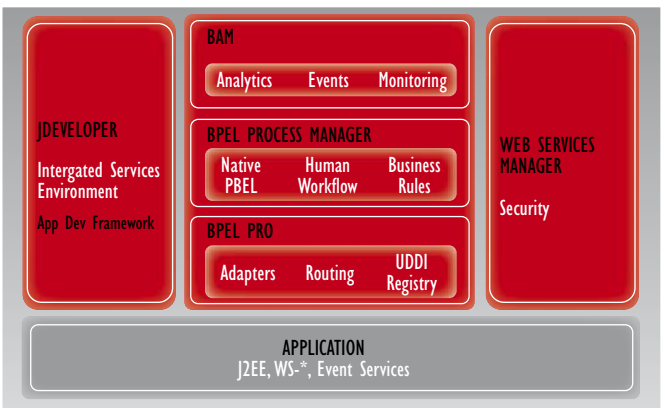
XML en SOA

In het afgelopen decennium heeft het gebruik van XML een enorme vlucht genomen. De belangrijkste reden hiervoor is dat XML een industriebrede standaard is; ondersteund door alle grote spelers, onder wie Microsoft, Oracle en IBM. Hierdoor heeft XML zich ontwikkeld tot het de facto platform- en technologieonafhankelijk uitwisselingsformaat, zijn er vele technische- en branchespecifieke berichtstandaarden ontwikkeld (XBRL, ebXML, SVG) en is een grote diversiteit aan ondersteunende tooling zoals parsers, readers, pretty-printers, transformators en API's beschikbaar. Naast het opslaan van configuraties wordt XML vooral gebruikt als neutraal uitwisselingsformaat tussen heterogene – waarvoor het de grootste toegevoegde waarde heeft – en ook homogene platformen. Dit wordt verder gestimuleerd door de opkomst van Service Oriented Architecture (SOA). Vanuit technisch oogpunt is



Figuur 1. XSL Map in JDeveloper

standaardisatie een belangrijke voorwaarde en drijfveer bij SOA. Dat heeft er in geresulteerd dat XML door zo'n beetje alle SOA-platformen gebruikt wordt als formaat voor informatie- en berichtstromen. Dit geldt ook voor het Oracle SOA-platform. Zowel BPEL Process Manager (BPEL PM) als Enterprise Service Bus (ESB) – beide onderdeel van het



Figuur 2. Oracle SOA-platform (image courtesy of Oracle Technology Network)

Oracle SOA-platform – gebruiken XML als formaat voor gegevensuitwisseling.

Binnen dit platform is BPEL PM bedoeld als tool om (mogelijk langlopende) bedrijfsprocessen uit te voeren. Hierbij gaat het vooral om de implementatie van proces- en applicatielogica. ESB is meer gericht op het implementeren en aanbieden van (technische) services binnen een SOA; getypeerd door vele kortlopende transacties. ESB biedt met name functionaliteit voor datatransformatie, routering, messaging, virtualisatie en het aanbieden van adapters voor verschillende systemen en technologieën. Het gebruik van XML-transformaties zal dus met name interessant zijn in combinatie met ESB. Let wel: de toepassing van XML kan ook doorslaan. Denk bijvoorbeeld aan het inladen van bulkdata via XML, waarbij de 'verbosity[1]'(1) van XML parten speelt. In dit opzicht wordt het overdadige gebruik van XML ook wel eens getypeerd als "XML-hell". XML is nu eenmaal net als SOA geen universele silver-bullet voor elk probleem.

Inleiding op XML-transformaties

Voor diegenen die nog niet met XML-transformaties gewerkt hebben, volgt hier een korte inleiding. XSLT staat voor eXtensible Stylesheet Language Transformations en is onderdeel van de XSL-standaard. XSL wordt gebruikt om XML-documenten te transformeren naar – met name – XML in een andere structuur. Het is echter ook mogelijk om transformaties naar andere formaten zoals HTML, PDF en CSV te definiëren. Hieronder staat een eenvoudig XSL-document. Als deze transformatie wordt toegepast op het XML-fragment `<rente>150</rente>` resulteert dat in `<interest>150<interest>`.

```
<xsl:stylesheet version="1.0"
          xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <interest><xsl:value-of select="rente"/></interest>
</xsl:template>
</xsl:stylesheet>
```

Om transformaties te definiëren zijn verschillende functies en navigatieconstructies beschikbaar in de XML-standaarden. Denk hierbij aan dataconversies, wiskundige functies, het itereren over elementen en attributen in een XML-document, enzovoort. Al deze constructies zijn te vinden op de website van het World Wide Web Consortium, die deze standaard beheert; zie: http://www.w3.org/TR/xslt. Vele belanghebbenden, waaronder Oracle, zijn lid van het W3C.

Case

In dit artikel wordt een Java-methode geïmplementeerd die een bankrekeningidentificatie omzet naar een bijbehorend IBAN (International Bank Account Number). Vervolgens zal deze methode als XSLT-extensie geregistreerd worden en in een XML-transformatie gebruikt worden. IBAN is een gestandaardiseerde structuur voor bankrekeningnummers die in de meeste Europese landen geldig is. Hierdoor zijn rekeningnummers uniek identificeerbaar over landgrenzen heen en wordt grensoverschrijdend betalingsverkeer sneller en efficiënter. IBAN is geregeld in het nieuws; laatst bijvoorbeeld bij de officiële start van de Single Euro Payments Area (SEPA) die verdere integratie van de Europese betalingsmarkt tot doel heeft.

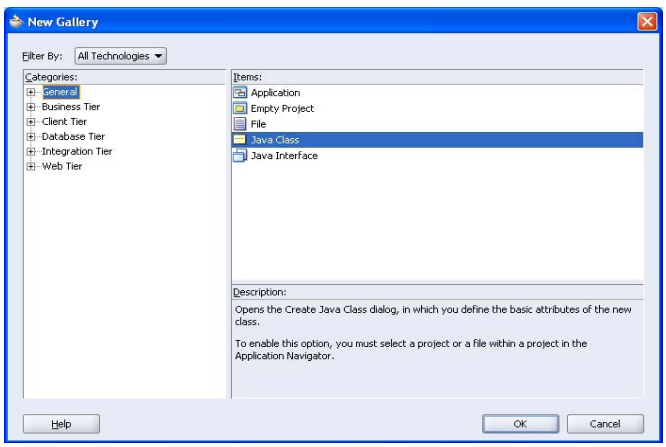
Installatie en configuratieomgeving

Download en installeer Oracle JDeveloper 10g Release 3 (10.1.3.1) of hoger. Dit product, inclusief installatie- en configuratie-instructies, is te vinden op Oracle Technology Network (OTN); zie: http://www.oracle.com/technology/software/products/jdev/index.html.

Tutorial

In het vervolg van dit artikel wordt stap voor stap uitgelegd hoe XSLT-extensies gecreëerd kunnen worden in JDeveloper.

1. Maak een nieuwe applicatie aan in JDeveloper via de menukeuze **File _ New…**. Dit opent het **New Gallery**-venster. Kies **General _ Application**.
2. Vul **XsltDemo** in als **Application Name**. Accepteer de overige standaard waarden en selecteer **OK**.
3. Het **Create Project**-venster verschijnt. Vul **XsltFunctions** in als **Project Name** en selecteer **OK**.
4. Creëer een nieuwe Java-klasse in het **XsltFunctions**-project. Klik met rechts op het project in de **Applications Navigator** en selecteer **New…**. Kies **General _ Java Class** in het **New Gallery**-venster.



Figuur 3. New Gallery-venster

5. Het **Create Java Class**-venster verschijnt. Vul de volgende waarden in:

a. Name: **XsltFunctions**
b. Package: **nl.optimize**
c. Deselecteer **Generate Default Constructor** aangezien XSLT-extensies als static methoden geïmplementeerd dienen te worden.
d. Accepteer de overige standaard waarden en selecteer **OK**.

We hebben een applicatie, project en Java-klasse aangemaakt. We gaan verder met de implementatie.

1. Open de zojuist aangemaakte Java-klasse **XsltFunctions**.
2. Maak een methode aan met de volgende signatuur:
   `public static String convertToIBAN(String landCode, String rekeningId)`

Deze methode dient het IBAN te retourneren voor de meegegeven landcode en rekeningidentificatie. Hierbij dient de bankcode onderdeel te zijn van de rekeningidentificatie. Een voorbeeld van geldige input is `NL` als land en `PSTB0001234567` als rekeningidentificatie. De functie dient `NL69PSTB0001234567` te retourneren.
Het gaat hierbij vooral om de berekening van het controlegetal; `69` in ons voorbeeld. Deze berekening wordt later verder uitgewerkt.

Dit voorbeeld is een versimpeling van de werkelijkheid.
3. Kopieer de code uit onderstaand tekstvak in de Java methode:

```
/**
 * Berekent en retourneert het IBAN behorend bij het meegegeven land en
 * rekening.
 *
 * @param landCode Twee-letterige landcode, bijvoorbeeld "NL".
 * @param rekeningId Rekeningidentificatie, bijvoorbeeld het imaginair
 * Nederlands postbanknummer PSTB0001234567. Deze identificatie is voor
 * Nederlandse rekeningnummers een 4-letterige bankcode, gevolgd door een
 * 10-cijferig rekeningnummer.
 *
 * @return Het IBAN van meegegeven land en rekeningidentificatie.
 */
public static String convertToIBAN(String landCode, String rekeningId) {

    StringBuffer iban = new StringBuffer();

    iban.append(landCode);
    iban.append(bepaalControleGetal(landCode, rekeningId));
    iban.append(rekeningId);

    return iban.toString();
```

Deze eenvoudige methode bepaalt het controlegetal en retourneert het bijbehorende IBAN. Om het controlegetal te berekenen, implementeren we vervolgens een nieuwe methode.

4. Maak een methode aan met de volgende signatuur:
   `private static long bepaalControleGetal(String landCode, String rekeningId)`

Omdat deze methode niet als XSLT-extensie beschikbaar wordt gemaakt, is deze `private`.
5. Kopieer de code uit onderstaand tekstvak in de Java-methode:

```
/**
 * Berekent en retourneert het IBAN controlegetal behorend bij het
 * meegegeven land en rekeningidentificatie.
 *
 * @param landCode Twee-letterige landcode, bijvoorbeeld "NL".
 * @param rekeningId Rekeningidentificatie, bijvoorbeeld het imaginair
 * Nederlands postbanknummer PSTB0001234567. Deze identificatie is voor
 * Nederlandse rekeningnummers een 4-letterige bankcode, gevolgd door een
 * 10-cijferig rekeningnummer.
 *
 * @return Het IBAN controlegetal.
 */
private static long bepaalControleGetal(
        String landCode, String rekeningId) {

    StringBuffer buffer = new StringBuffer();

    buffer.append(rekeningId);
    buffer.append(landCode);

    StringBuffer controleReeks = new StringBuffer();

    // Itereer over buffer en converteer letters naar bijbehorende cijfers
    for (int index = 0; index < buffer.length(); index ++) {

        controleReeks.append(
            Character.getNumericValue(buffer.charAt(index)));
    }

    // Voeg 2 nullen toe aan het einde
    controleReeks.append("00");

    // Bepaal controlegetal via: 98 - (controlereeks mod 97)
    return (98 -
        (new BigInteger(controleReeks.toString()).mod(
           BigInteger.valueOf(97))).intValue());
}
```

Deze methode implementeert de berekening van het IBAN-controlegetal via onderstaand algoritme (zie: http://nl.wikipedia.org/wiki/International_Bank_Account_Number#Controlegetal):

a. Neem de rekeningidentificatie
b. Plaats de landcode erachter
c. Vervang alle letters door hun positie in het Romeinse alfabet, vermeerderd met 9 (A=10, B=11, ..., Z=35)
d. Voeg twee nullen toe aan het einde
e. Bepaal de restwaarde na deling van het verkregen getal door 97
f. Haal deze rest van 98 af om het controlegetal te krijgen.

6. Voeg een `main` methode toe om deze klasse locaal te testen:

```
/**
 * Main methode om deze klasse locaal te kunnen testen.
 *
 * @param args Argumenten.
 */
public static void main(String[] args) {

    // Voorbeeld argumenten: "NL", "PSTB0001234567"
    System.out.println(convertToIBAN(args[0], args[1]));
}
```

Voer de volgende stappen uit om deze klasse te testen:

1. Klik met rechts op het **XsltFunctions**-project in de **Applications Navigator** en selecteer **Project Properties…**. Dit opent het **Project Properties**-venster.
2. Selecteer **Run/Debug**. Selecteer **Default** in de lijst van **Run Configurations** en klik op **Edit…**. Dit opent het **Edit Run Configuration**-venster.
3. Geef een landcode en rekeningidentificatie op als **Program Arguments**; bijvoorbeeld: `NL PSTB0001234567`. Selecteer **OK** en nogmaals **OK**.
4. Klik met rechts op de **XsltFunctions**-klasse in de **Applications Navigator** en selecteer **Run…** om de klasse te testen.
5. Het **Log**-venster toont de uitkomst. Dit zou `NL69PSTB0001234567` moeten zijn.

Controleer eventueel andere combinaties door de uitkomst van bovenstaande klasse te vergelijken met de IBAN-checker op www.ibannl.org.

Registeren als XSLT-extensie

Nu de functionaliteit geïmplementeerd en getest is, gaan we deze beschikbaar maken als XSLT-extensie. Hiervoor dienen we deze klasse als JDeveloper-extensie te packagen en de XSLT-functie te registeren.

Een extensie is een toevoeging op de basisfunctionaliteit van JDeveloper; denk bijvoorbeeld aan integratie met Subversion, tooling om Java-code te analyseren. In de Java-gemeenschap bestaat een initiatief (JSR-198) om de ontwikkeling van extensies voor Java IDE's (Integrated Development Environments) via een gedeelde API te standaardiseren. Meer hierover is te vinden in de `[Oracle JDeveloper]/jdev/doc/extension` directory en op OTN: http://www.oracle.com/technology/products/jdev/howtos/1013/extension/extension.html.

Aanmaken JDeveloper-extensie

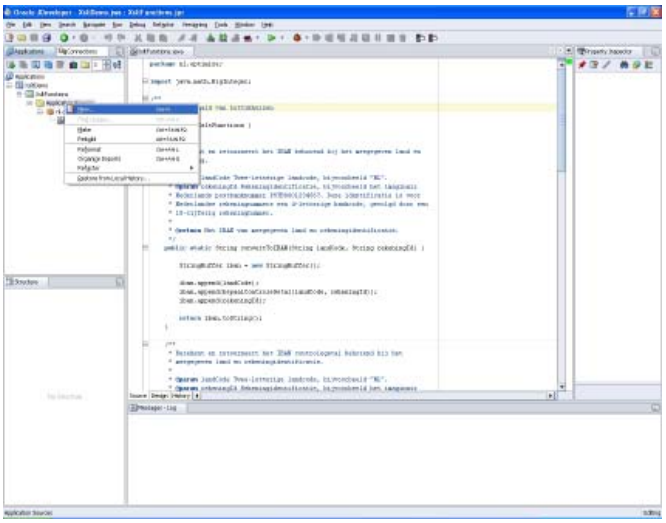
Voer de volgende stappen uit om de klasse als JDeveloper-extensie beschikbaar te stellen.

1. Selecteer de **Application Sources**-node onder het **XsltFunctions**-project in de **Applications Navigator**. Maak een JDeveloper-extensieconfiguratie aan via menukeuze **File _ New…**. Kies **General _ XML _ Application _ XML Document** in het **New Gallery**-venster.
2. Vul `extension.xml` als **File Name** in en voeg `META-INF` toe achter `src` in de directory-naam. Selecteer **OK**.
3. Kopieer het XML-document uit onderstaand tekstvak in het XML-document:

```
<?xml version="1.0" encoding="UTF-8" ?>
<extension version="1.0" esdk-version="1.0" id="XsltFunctions"
           xmlns="http://jcp.org/jsr/198/extension-manifest">
  <name>IBAN XSLT Extensie</name>
  <owner>Optimize</owner>
</extension>
```

Dit configuratiebestand specificeert een extensie volgens Java-standaard JSR 198.

4. Selecteer het **XsltFunctions**-project in de **Applications Navigator**. Maak een deployment-profile aan om het project te packagen via menukeuze **File _ New…**. Kies **General _ Deployment Profiles _ JAR File** in het **New Gallery**-venster.
5. Vul `XsltFunctions_JDeveloper_Extension` als **Deployment Profile Name** in. Selecteer **OK**. Het **JAR Deployment Profile Properties**-venster verschijnt. Selecteer nogmaals **OK**. Voeg `XsltFunctions.1.0.jar` toe achter de directory-naam in **JAR File**.
6. Package het project als JAR-bestand. Klik met rechts op het nieuw aangemaakte deployment-profile en selecteer **Deploy to JAR file**. In het **Log**-venster verschijnt de melding **Deployment finished**.



Figuur 4. Applications Navigator in JDeveloper

7. Kopieer het bestand `XsltFunctions.1.0.jar` uit de `deploy` directory van het project naar de `[Oracle JDeveloper]/jdev/extensions` directory.
8. Herstart JDeveloper. Mocht de extensie niet geldig zijn, dan verschijnt de volgende melding in het **Log**-venster: `Severe(0,0): file:/[JDeveloper_Installatie_Directory]/jdev/extensions/XsltFunctions.1.0.jar does not contain an extension manifest.`

Registreren XSLT-extensie

Voer de volgende stappen uit om de klasse als XSLT-extensie te registreren en deze in de XSL Map te kunnen gebruiken.

1. Selecteer het **XsltFunctions**-project in de **Applications Navigator**. Maak een XSLT-extensieconfiguratie aan via menukeuze **File _ New…**. Kies **General _ XML _ Application _ XML Document** in het **New Gallery**-venster.
2. Vul `XsltFunctions.xml` in als **File Name** in. Selecteer **OK**.
3. Kopieer het XML-document uit onderstaand tekstvak in het XML-document:

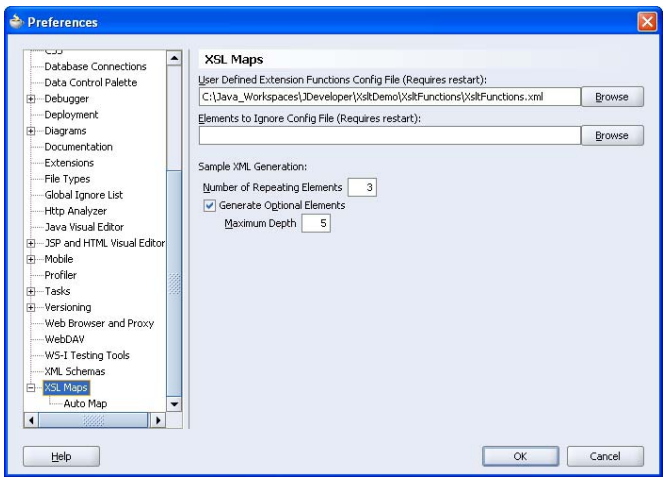
```
<?xml version="1.0" encoding="UTF-8"?>
<extension-functions>
  <functions xmlns:optimize="http://www.oracle.com/XSL/Transform/java/
nl.optimize.XsltFunctions">
    <function name="optimize:convertToIBAN" as="string">
      <param name="landCode" as="string"/>
      <param name="rekeningId" as="string"/>
    </function>
  </functions>
</extension-functions>
```

Let op: de namespace waarin deze functies gedefinieerd worden, dient altijd te beginnen met `http://www.oracle.com/XSL/Transform/java/` gevolgd door de volledige package- en klassenaam van de Java-klasse die de functies implementeert.

4. Selecteer menukeuze **Tools _ Preferences…**. Dit opent het **Preferences**-venster.
5. Selecteer **XSL Maps**. Klik op **Browse** naast het tekstvak **User Defined Extension Functions Config File** en selecteer het `XsltFunctions.xml` document. Selecteer **OK**.
6. Herstart JDeveloper.

Aanmaken transformatie via de XSL Map

Vervolgens kan de XSLT-extensie gebruikt worden in de XSL Map. Deze grafische tool baseert een transformatie op een bron- en doeldatastructuur die beide als XML Schema gedefinieerd zijn. Voer de volgende stappen uit om een bron- en doel-XML Schema en de daadwerkelijke transformatie te definiëren.



Figuur 5. Registratie van extensiefunctie

1. Selecteer het **XsltFunctions**-project in de **Applications Navigator**. Maak een bron- en doel-XML Schema aan via menukeuze **File _ New…**. Kies **General _ XML _ Application _ XML Schema** in het **New Gallery**-venster.
2. Vul `XsltDemoSource.xsd` als **File Name** in. Voeg `public_html` toe achter de directory-naam. Selecteer **OK**. Dit opent een voorbeeld XML Schema.
3. Selecteer het tabblad **Source** en vervang de bestaande code door het XML-document in onderstaand tekstvak.

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns="http://www.optimize.nl/xsltDemo/source"
            targetNamespace="http://www.optimize.nl/xsltDemo/source"
            elementFormDefault="qualified">
  <xsd:element name="input">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="land" type="xsd:string"/>
        <xsd:element name="rekeningId" type="xsd:string"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

4. Herhaal stappen 1 tot en met 3. Geef dit keer als bestandsnaam `XsltDemoTarget.xsd` op en gebruik het XML-document uit onderstaand tekstvak.

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
```

```
            xmlns="http://www.optimize.nl/xsltDemo/source"
            targetNamespace="http://www.optimize.nl/xsltDemo/target"
            elementFormDefault="qualified">
  <xsd:element name="output">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="ibanRekeningNummer" type="xsd:string"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Gebruik maken van de XSLT-extensie in de XSL Map

De functie kan nu gebruikt worden in de XSL Map om een XML-transformatie te definiëren.

1. Selecteer het **XsltFunctions**-project in de **Applications Navigator**. Definieer een XML-transformatie in XSL Map via menukeuze **File _ New…**. Kies **General _ XML _ Application _ XSL Map** in het **New Gallery**-venster.
2. Vul `XsltDemoTransformatie.xsl` als **File Name** in. Navigeer in het **Source**-panel naar **Available Schemas _ Project Schema Files _ XsltDemoSource.xsd _ input**. Navigeer in het **Target**-panel naar **Available Schemas _ Project Schema Files _ XsltDemoTarget.xsd _ output**. Selecteer **OK**. Dit opent de nu nog lege transformatie in de XSL Map. Aan de linkerkant is de bronstructuur zichtbaar en aan de rechterkant de doelstructuur. De transformatie tussen beide ontbreekt nog.

Alle beschikbare transformatie- en navigatiefuncties zijn te vinden in het **Component Palette** en gegroepeerd in verschillende categorieën. Deze functies bevatten niet alleen een groot aantal standaard XSL(T) en XPath-constructies, maar ook Oracle-eigen functies zoals het ophalen van ESB Domain Value Maps (DVM), het gebruiken van sequence waarden, enzovoort.
3. Bekijk de door ons gedefinieerde extensiefuncties door in het **Component Palette** de categorie **User Defined Extension Functions** te selecteren. Daarin staat de functie `convertToIBAN`. Drag-en-drop deze functie vanaf het **Component Palette** naar de XSL Map tussen de bron- en doelstructuur in.

4. Verbind de inputparameters **land** en **rekeningId** door beide naar de linker connector van de functie in het midden van de XSL Map te slepen. Controleer de volgorde van de parameters door te dubbelklikken op de functie. Dit opent het **Edit Function**-venster waarin de volgorde van parameters gecontroleerd kan worden.
5. Verbind nu de uitkomst van de functie met de outputparameter **ibanRekeningNummer** door de rechter connector van de functie naar de outputparameter te slepen.

Bewaar de transformatie via menukeuze **Save**. De mapping wordt opgeslagen als `XsltDemoTransformatie.xsl` bestand.

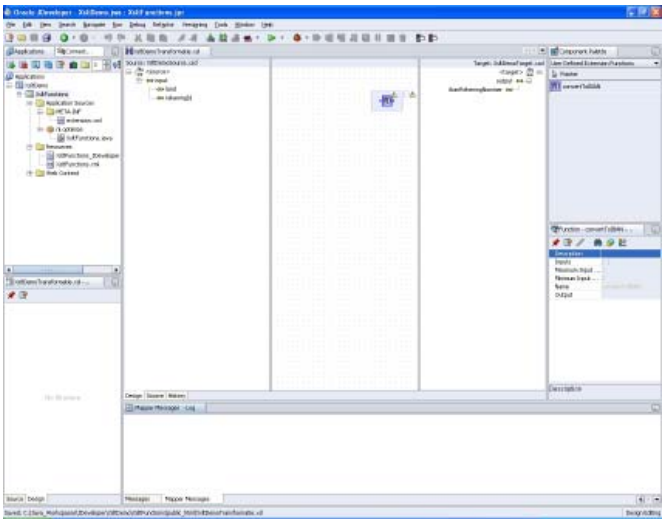
Testen van de XSL Map

JDeveloper helpt ons een handje met het testen van de XML-transformatie.

1. Selecteer de **Application Sources**-node onder het **XsltFunctions**-project in de **Applications Navigator**. Maak een voorbeeldinput-XML-bestand aan via menukeuze **File _ New…**. Kies **General _ XML _ Application _ XML Document** in het **New Gallery**-venster.
2. Vul `XsltDemoTransformatie-Source.xml` als **File Name** in en voeg `public_html` toe achter de directory-naam. Selecteer **OK**.
3. Kopieer het XML-document uit het onderstaand tekstvak in het XML-document:

```
<?xml version="1.0" encoding="UTF-8" ?>
<input xmlns="http://www.optimize.nl/xsltDemo/source">
   <land>NL</land>
   <rekeningId>PSTB0001234567</rekeningId>
</input>
```

4. Creëer een test. Klik met rechts op het `XsltDemoTransformatie.xsl` bestand in de **Applications Navigator**. Selecteer **Test**. Dit opent het **Test XSL Map**-venster.
5. Selecteer het zojuist aangemaakte document als **Source XML File**. Deselecteer de optie **Generate Source XML File**. Accepteer de overige standaard waarden en selecteer **OK**. Dit zorgt ervoor dat JDeveloper de transformatie uitvoert en het resultaat opslaat als XML-document met de



Figuur 6. XSL Map en User Defined Extension Functions

naam `XsltDemoTransformatie-Target.xml`. Open dit bestand – als het al niet getoond wordt – en controleer of het voldoet aan het doel-XML Schema en de waarde `NL69PSTB0001234567` bevat.

Naast het kunnen testen van een XML-transformatie is er nog een fraaie en vooral nuttige feature in JDeveloper beschikbaar, namelijk het kunnen debuggen van XML-transformaties. Zie de volgende demo op OTN voor meer informatie hierover: http://www.oracle.com/technology/products/jdev/101/viewlets/101/xsltdebug_viewlet_swf.html.

Toepasbaarheid

Het gebruik van XSLT-extensies is vrij beperkt in de meeste 'standaard' Java/JEE-applicaties. Zoals in de inleiding is aangegeven, zal de toepassing ervan binnen SOA vele malen groter en een stuk interessanter zijn. Aangezien ESB-projecten design-time gebruik maken van de XSL Map voor transformaties, is de registratie van extensies, zoals hierboven beschreven, genoeg om deze functionaliteit te gebruiken in de ESB Designer van JDeveloper. Om deze extensies run-time te laten werken, dienen deze beschikbaar gemaakt te worden in de Oracle SOA Suite. Dit kan door de gepackagede extensies als shared-library te definiëren of deze in de `[Oracle SOA Suite]/j2ee/[SOA Suite OC4J container]/applib` directory te plaatsen.

Waarom XSLT-extensies gebruiken?

Aangezien er al vele standaard en Oracle-specifieke transformatiefuncties bestaan, rijst misschien – en terecht – de vraag welke toegevoegde waarde XSLT-extensies hebben? Eén van de beperkingen van 'standaard' XSLT is dat met de beschikbare set aan transformaties niet alle gewenste functionaliteit te implementeren valt. Dit komt vanwege het karakter van XSLT. Het is immers een declaratieve templating-taal en niet bedoeld als imperatieve programmeertaal. Dit soort 'onmogelijke' functionaliteit kan wel met Java geïmplementeerd worden. Bovendien kan functionaliteit veelal wel bereikt worden door een aaneenrijging van bestaande XSLT-constructies ('chaining' van functies). Doordat XML echter 'verbose' is, vermindert de leesbaarheid en is de kans op fouten in dergelijke transformaties vrij snel groter. In zulke gevallen kan het gebruik van één zelfgedefinieerde extensie te prefereren zijn. Een laatste voor de hand liggend voordeel is de mogelijke snelheidswinst – met name run-time, maar mogelijk ook design-time – van Java boven 'native' XSLT (gecompileerde code versus interpretatie). Een nadeel van het gebruik van XSLT-extensies is dat Java-code, in tegenstelling tot XSLT, gecompileerd moet worden. Verder vereist het gebruik van XSLT-extensies additionele configuratie en leidt de mix van verschillende technologieën tot grotere complexiteit tijdens design, implementatie en testen.

Bovendien zijn alleen de volgende – beperkte set aan – datatypen toegestaan als attribuuttypen voor XSLT-extensies: `boolean`, `number`, `node-set` en `tree`. Dit zijn er een stuk minder dan die door de XML Schema standaard worden voorgeschreven.
Als laatste kan in JDeveloper maar één extensieconfiguratie tegelijkertijd opgegeven worden. Een mogelijkheid om dit probleem te omzeilen, is het gebruik van een façade die delegeert aan meerdere helpers. Functies kunnen zo gegroepeerd worden in onderliggende Java-klassen. Over de XSL Map dient nog opgemerkt te worden – onafhankelijk van het feit of XSLT-extensies gebruikt worden – dat niet de gehele XSL(T) en XPath standaard ondersteund wordt. Voorbeelden van niet ondersteunde constructies zijn `XSL:element` en het XML Schema type `any`. Bij gebruik van een dergelijke constructie is de grafische mapping niet meer zichtbaar en kan het XSLT-document alleen rechtstreeks bewerkt worden. Houdt deze voor- en nadelen in het oog bij het afwegen van het gebruik van XSLT-extensies.

Conclusie

Dit artikel laat zien hoe Java-methoden in JDeveloper als XSLT-extensies geregistreerd, gebruikt en getest kunnen worden en wat de voor- en nadelen er van zijn. Het voornaamste toepassingsgebied van deze extensies is op het Oracle SOA-platform; met name ESB en BPEL PM.

Links naar gerelateerde informatie

- Voor meer informatie over Oracle SOA zie: http://www.oracle.com/technology/soa/index.html.
- Oracle SOA Suite bevat een XSLT extensie demo; zie [Oracle SOA Suite]/bpel/samples/demos/XSLMapper/ExtensionFunctions.
- XSL Map extensie functies zijn niet hetzelfde als zelfgedefinieerde BPEL PM XPath functies; zie hiervoor de BPEL PM Developers Guide op OTN.
- Het artikel "Invoice Processing in a Service-Oriented Environment" op OTN geeft een voorbeeld hoe en op welke wijze XML transformaties in een ESB flow ingezet kunnen worden.

Ronald van Luttikhuizen (ronald.vanluttikhuizen@approach-alliance.nl) is als architect werkzaam bij Approach Alliance (www.approach-alliance.com), een ICT-dienstverlener gespecialiseerd in SOA en Business Intelligence. De Approach Alliance weblog is te vinden op: http://www.approach-alliance.nl/index.php?option=com_jd-wp&Itemid=52.

(1) Verbosity – XML is een tekstformaat en gebruikt omschrijvingen, zogenaamde tags, om informatie te beschrijven en structureren. Hierdoor zijn XML-data over het algemeen vele malen groter dan dezelfde data in binair formaat.

Artikelen met praktische informatie, geschreven door en bestemd voor Oracle-professionals vindt u in het Online Archief van Array Publications. Vaktijdschriften als Database Magazine, Software Release en Java Magazine hebben hun artikelenarchief online gezet. Met een heldere zoekstructuur vindt u snel wat u zoekt op www.optimize.nl.

KPN rondt implementatie Oracle Application Architecture succesvol af

KPN heeft de implementatie van Oracle Application Integration Architecture (AIA) for Communications succesvol afgerond. Dit zorgt voor de koppeling tussen Siebel CRM 8.0 en Oracle Communications Billing and Revenue Management. Bovendien is het mogelijk bedrijfsprocessen in te richten die beschikbaar zijn over alle applicaties heen. De implementatie is onderdeel van KPN's inspanningen om zijn IT-systemen te vereenvoudigen en aan te passen om zo klanten beter van dienst te kunnen zijn. De implementatie van Oracle AIA for Communications zorgt voor een snellere introductie van nieuwe producten en diensten door synchronisatie van productdetails tussen CRM en Billing. De inspanningen voor marktintroductie is zelfs afgenomen met 10 tot 20 procent. Bovendien is de datakwaliteit van klant- en ordergegevens verbeterd door automatische synchronisatie tussen facturering- en klantinformatiesystemen. Dit heeft ook geleid tot een betere klanttevredenheid doordat medewerkers van het callcenter nu beschikken over accurate en consistente gegevens.

Technology Preview Fusion Middleware 11g

Oracle maakte tijdens de eigen algemene sessie van JavaOne op woensdag de verkrijgbaarheid van de technology-preview van Fusion Middleware 11g bekend (TP 4). De productieversie van Fusion Middelware 11g zou nog dit jaar vrijgegeven moeten worden, al twijfelen sommigen daaraan. In de preview onder meer: JDeveloper 11g IDE met ingebouwde ondersteuning voor webontwikkeling met Oracle ADF en onder meer de OC4J-container, de Oracle 11g SOA Suite met de Mediator en BPEL Service Engines, de Complex Event Processor en de BAM Server, de Rules Engine en de Human Workflow Service. Voor wat betreft JDeveloper 11g is de preview vrij compleet. Oracle zelf noemt het een mijlpaal in het proces van applicatieontwikkeling. Schromelijk overdreven uiteraard, maar voor Oracle-ontwikkelaars waarschijnlijk wel een belangrijke verbetering.