Enterprise

SOA is de architectuurstijl die er naar streeft componenten via services aan elkaar te koppelen. Maar op welk communicatieprotocol (Web Services, JMS, EJB) baseer je de implementatie van de services? En hoe houd je overzicht op de aaneenschakeling van die componenten? SCA biedt antwoord op deze vragen.

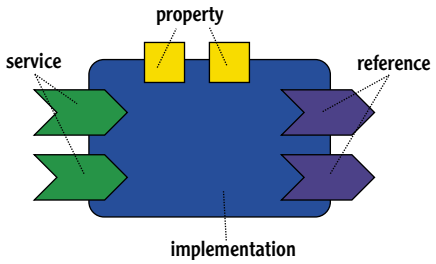
Service Component Architecture

Rond 2010 gebruikt een kwart van de nieuwbouwapplicaties SCA

Zoals wegen naar huizen leiden zo leiden verbindingen in het IT-landschap naar componenten. In de loop van de tijd zijn de wegen geëvolueerd van karrensporen naar ZOAB en de huizen van baksteen naar betonnen prefab. Maar er is iets vreemds aan de hand als we naar het IT-landschap kijken. Hoewel we een huis niet aanpassen als we een weg vervangen door een ander type weg, passen we een component vaak wel aan als we het met een ander type verbinding ontsluiten. SCA wil voorkomen dat we componenten moeten aanpassen, omdat de verbinding daartussen verandert. Hiervoor moeten we het koppelvlak tussen een component en de verbin-

Ronald Streekstra
is enterprise architect bij
Atos Origin
Andy Verberne
is java consultant bij
Atos Origin

ding standaardiseren. SCA definieert hoe een component gebouwd moet worden om gebruik te kunnen maken van verschillende typen verbindingen. Hierbij abstraheert SCA de bekende verbindingimplementaties als JMS, EJB, Web services. Daarnaast maakt SCA het mogelijk een plattegrond te definiëren waarin de verbindingen en componenten in een IT-landschap vastgelegd zijn. Belangrijk is dat deze definitie niet alleen een mooi plaatje oplevert, maar ook de in runtime gebruikte configuratie is. Hiermee lopen beschrijving en werkelijkheid synchroon.



Afbeelding 1. SCA component

Service Component Architecture (SCA)

In SCA is de implementatie van een servicecomponent losgekoppeld van het gebruikte communicatieprotocol. Bovendien kunnen componenten expliciet geassembleerd worden in een composite, waarbij het gewenste communicatieprotocol per koppeling kan worden geconfigureerd. Een composite is een bundeling van aan elkaar gekoppelde componenten en kan weer gebruikt worden als servicecomponent binnen een andere composite. Een SCA-applicatie kan zo worden opgebouwd door het assembleren van verschillende componenten en composites.

De businesslogica van een component kan met verschillende talen of frameworks worden gerealiseerd zoals Java, Spring, C++, Cobol, BPEL of verscheidene scripting-talen. Een component biedt zijn logica aan door middel van één of meer *services*. Dit gebeurt meestal op basis van WSDL. Wanneer de logica afhankelijk is van andere services, dan worden deze aangeroepen via een *reference*. Met een *property* kunnen variabele eigenschappen van een specifieke component worden geconfigureerd. Dit alles wordt geïllustreerd in afbeelding 2.

Services, references en properties representeren de configureerbare aspecten van een component, we noemen dit ook wel het *component type*. Deze aspecten kunnen bij sommige implementatietalen via introspectie worden achterhaald. In Java kan dit bijvoorbeeld door annotaties te gebruiken. Mocht dit niet door de taal worden ondersteund, dan worden de services, references en properties van een implementatie expliciet beschreven in een componentType-file.

Een composite, een bundeling van componenten, wordt beschreven in Service Component Definition Language (SCDL, uitgesproken als skiddle). Dit is een gestandaardiseerd XML-schema. In SDCL worden de componenten gedeclareerd en geconfigureerd door properties een waarde te geven en references te verbinden met services. Door services, references en properties van componenten binnen een composite te *promoveren*, wordt een composite op zijn beurt weer een component. Afbeelding 2 illustreert de koppeling van componenten via *wires* en de promotie van services en references binnen een composite.

Het communicatieprotocol waarmee components met elkaar praten worden gespecificeerd met een *binding*. Enkele mogelijke voorbeelden voor binding zijn webservice, JMS, EJB Session Bean en SCA. Naast het te gebruiken protocol kan ook opgegeven worden of de uitwisseling van de gegevens betrouwbaar, veilig of transactioneel moet zijn. In SCA wordt dit geregeld door *policies*.

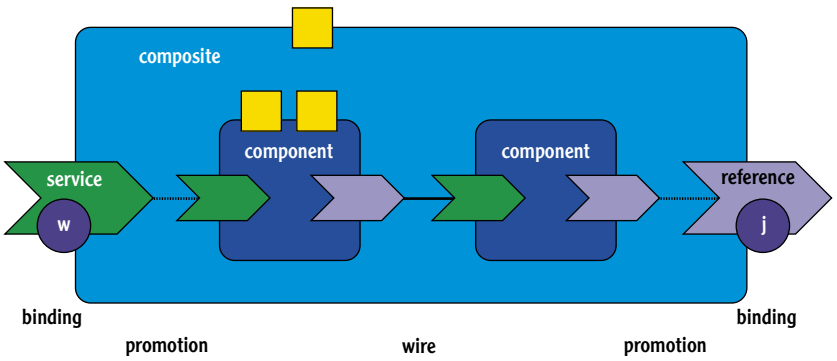
SCA voorbeeld

Tijd voor een voorbeeld om de bovenstaande theorie wat te duiden. Het voorbeeld bestaat uit drie componenten:

- BehandelAanvraagComponent
- BerekenPremieComponent
- BewaarPolisComponent

De eerste component behandelt de aanvraag van een verzekering en gebruikt daarbij de tweede component voor de berekening van de premie, en de derde component voor het bewaren van de polis.

Onafhankelijk van hoe deze componenten geïmplementeerd gaan worden, kunnen de services die ze aanbieden en references die ze gebruiken worden beschreven in een componentType-file. De interfaces zijn daarbij gespecificeerd in WSDL. De componentType-file voor het



Afbeelding 2. SCA Composite

BehandelAanvraagComponent ziet er als volgt uit; de details van de WSDL-files zelf zijn weggelaten.

```
<?xml version="1.0" encoding="UTF-8"?>
<componentType xmlns=
       "http://www.osoa.org/xmlns/sca/1.0">

  <service name="BehandelAanvraagService">
    <interface.wsdl interface="…"/>
  </service>

  <reference name="berekenPremieReference">
    <interface.wsdl interface="…"/>
  </reference>

  <reference name="bewaarPolisReference">
    <interface.wsdl interface="…"/>
  </reference>

</componentType>
```

De componenten kunnen vervolgens in een composite worden samengevoegd. In dit voorbeeld worden de eerste twee componenten gebundeld in VerzekeringAanvraagComposite en de derde component in een aparte composite die in dit voorbeeld niet verder wordt toegelicht. Afbeelding 3 toont de VerzekeringAanvraagComposite. Afbeelding 3 toont hoe de BehandelAanvraagService (B) van de BehandelAanvraagComponent is gepromoveerd als service (A) van de VerzekeringAanvraagComposite. Verder zien we hoe de berekenPremieReference (C) van de BehandelAanvraagComponent via een wire is verbonden met de BerekenPremieService (D) van de BerekenPremieComponent. Ten slotte is te zien hoe de bewaarPolisReference (E) is gepromoveerd als BewaarPolisReference (F) van de VerzekeringAanvraagComposite.

Hieronder staat de SCDL-file van het voorbeeld.

```
<?xml version="1.0" encoding="UTF-8"?>
<composite
  xmlns="http://www.osoa.org/xmlns/sca/1.0"
  name="VerzekeringAanvraagComposite">

<component name="BehandelAanvraagComponent">
  <implementation.bpel
    process="BehandelAanvraag"/>
</component>

<component name="BerekenPremieComponent">
  <implementation.script
    script="BerekenPremie.js"/>
</component>

<service
  name="VerzekeringAanvraagService"
  promote="BehandelAanvraagComponent/
           BehandelAanvraagService"/>

<reference
  name="BewaarPolisReference"
  promote="BehandelAanvraagComponent/
           bewaarPolisReference"/>

<wire
  source="BehandelAanvraagComponent/
          berekenPremieReference"
  target="BerekenPremieComponent/
          BerekenPremieService"/>

</composite>
```

Uit de SCDL-file valt af te lezen dat de BehandelAanvraagComponent is geïmplementeerd als een BPEL-proces en de BerekenPremiecomponent is geschreven in JavaScript.

SCA en Java

Wanneer als implementatie van de componenten is gekozen voor de taal Java, dan zijn de componentType-files niet meer nodig. Door gebruik te maken van annotaties kan de SCA run-time de services die een component biedt, en de references die hij gebruikt, via introspectie bepalen. Ook kunnen de interfaces van services en references via een Java-interface worden gespecificeerd in plaats van in WSDL. Onderstaande SCDL-snippet toont de aanpassing in de vorige SCDL-file om de implementatie van de componenten te veranderen in Java.

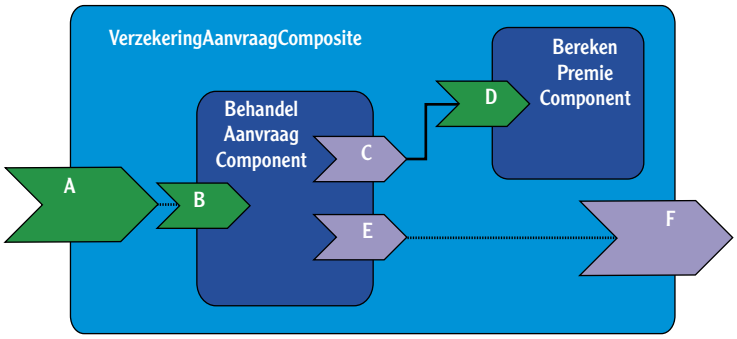
```
<component name="BehandelAanvraagComponent">
  <implementation.java
      class="com.x.BehandelAanvraagImpl"/>
</component>

<component name="BerekenPremieComponent">
  <implementation.java
      class="com.x.BerekenPremieImpl"/>
</component>
```

Aan het implementatietype wordt de class meegegeven die de implementatie verzorgd. De drie Java-interfaces en twee Java-implementaties van deze componenten zijn als volgt:

```
@Remotable
public interface BehandelAanvraagService {
  public Polis behandelAanvraag(
```

Afbeelding 3. SCA voorbeeld



```
                  Aanvraag aanvraag);
}

public interface BerekenPremieService {
  public double berekenPremie(
                   Aanvraag aanvraag);
}

@Remotable
public interface BewaarPolisService {
  public void bewaarPolis (Polis polis);
}

@Service(BehandelAanvraagService.class)
public class BehandelAanvraagImpl
  implements BehandelAanvraagService {

  @Reference
  protected BerekenPremieService
              berekenPremieReference;

  @Reference
  protected BewaarPolisService
              bewaarPolisReference;

  public Polis behandelAanvraag(
                  Aanvraag aanvraag) {
    …

    berekenPremieReference.
      berekenPremie(aanvraag);
    bewaarPolisReference.
      bewaarPolis(polis);
    …
  }
}

@Service(BerekenPremieService.class)
public class BerekenPremieImpl
  implements BerekenPremieService {

  public double berekenPremie(
                   Aanvraag aanvraag) {
    …
  }
}
```

De @Remotable tag geeft aan dat een service is ontworpen voor *remote* communicatie met pass by-value semantiek. Dit is vooral van toepassing voor coarse grained services. De @Service tag specificeert de services die een class aanbiedt. De references die een class gebruikt, worden geannoteerd met de @Reference annotatie.

SCA Bindings

In het voorbeeld is niet gespecificeerd via welk protocol de componenten aangeroepen worden. Het specificeren van het protocol gaat via een binding. De default binding is SCA binding. Dit is een door de leverancier gekozen protocol dat door de SCA run-time ingevuld kan worden. In ons voorbeeld roept de BehandelAanvraagComponent de BerekenPremieComponent aan via SCA binding.

Door de binding expliciet te specificeren kunnen we de VerzekeringAanvraagService aanbieden als een webservice. Ook kunnen we de aanroep van de BewaarPolisComponent via de BewaarPolisReference asynchroon laten plaatsvinden door JMS te specificeren als binding op de reference. Dit wordt geïllustreerd in onderstaande SCDL-snippet waarin de service en reference uit de eerder getoonde SCDL-file uitgebreid zijn met bindings.

```
<service
  name="VerzekeringAanvraagService"
  promote="BehandelAanvraagComponent/
           BehandelAanvraagService">
  <binding.ws uri="http://localhost:8080/..."/>
</service>

<reference
  name="BewaarPolisReference"
  promote="BehandelAanvraagComponent/
           bewaarPolisReference">
  <binding.jms uri="jms:BewaarPolisQueue?…"/>
</reference>
```

De binding tags worden voorzien van de nodige informatie met betrekking tot het gekozen protocol. Zo kan bij de binding.ws tag de URL worden opgeven waarmee de webservice kan worden aangeroepen. En moet bij de binding.jms tag de gebruikte JMS-provider en destination nog worden opgegeven.

SCA-producten

SCA bestaat in zijn eerste vorm vanaf 2005 en is in maart 2007 vrijgegeven in de 1.0-specificatie. SCA kan worden geïntegreerd in een run-time omgeving (application server of ESB) en visuele ondersteuning bieden voor het assembleren van composite applicaties (Eclipse). Voor dit artikel hebben we naar vijf partijen gekeken: Tuscany (open source), IBM, Oracle, BEA en JBoss. Hierbij viel de laatste snel af omdat JBoss geen ondersteuning biedt voor SCA.

Tuscany (http://incubator.apache.org/tuscany/) is een open source SCA-implementatie van Apache. Versie 1.0.1 van Tuscany is sinds november 2007 uit. Deze versie ondersteunt de volgende implementatie types en bindings.

Implementatie Types	Bindings
POJO	SCA
Spring Bean	Web Services
Scripting	EJB
(JavaScript, Groovy,	HTTP
Ruby, Python)	JSON-RPC
BPEL	RMI
OSGI	

Tuscany kan stand-alone draaien of geïntegreerd met een application server. Naast de Java-implementatie is er ook een C++-implementatie. De Java-versie is wat betreft stabiliteit in het afgelopen jaar sterk verbeterd, maar tijdens het onderzoek voor dit artikel zijn we nog wel bugs tegengekomen.

IBM WebSphere Process Server (WPS) 6.0 is eind 2005 uitgekomen met een SCA run-time gebaseerd op de SCA 0.9 specificaties. Deze run-time vormt, samen met de ondersteuning voor SDO en Common Event Infrastructure (CEI), de SOA-core van WPS en WebSphere ESB. In een volgende versie van WPS zal Tuscany gebruikt worden als SCA run-time en daarmee zal WPS dus ook de SCA 1.0-specificaties ondersteunen. Een voorproefje hiervan is al beschikbaar voor WAS 6.1 in de vorm van een Feature Pack for SOA. De ontwikkelomgeving voor WPS is WebSphere Integration Developer (WID) die gebaseerd is op Eclipse. WID bevat grafische ondersteuning in de vorm van een Assembly Model Editor voor het creëren van een composite waardoor de onderliggende SCDL-file wordt afgeschermd voor de ontwikkelaar.

Oracle toonde op de Java One 2007 al een preview van Oracle 11g producten met daarin SCA-ondersteuning. Daarin was onder meer procesintegratie en een visuele SCA-editor te zien. Afgelopen december heeft Oracle de eerste publieke bètarelease vrijgegeven van haar *Oracle 11g SOA Suite* – een SCA-container met ondersteuning en implementatie van ondermeer BPEL en een Enterprise Service Bus.

BEA heeft december vorig jaar de *SCA for WebLogic Server 10.3 Tech Preview* vrijgegeven. Dit is een SCA run-time voor de aankomende WebLogic Server 10.3 release. Deze run-time is gebaseerd op de *fabric3* open source SCA-implementatie. In de nabije toekomst ondersteunt BEA ook het assembleren van SCA-composites met een Eclipse plugin in BEA *Workshop*. Met de overname van BEA door Oracle heeft Oracle nu twee SCA-containers. Wat dat in de toekomst gaat betekenen is nog onzeker.

Overwegingen voor SCA

SCA is naar onze mening een aanwinst voor programmeurs en architecten die werkzaam zijn op het gebied van applicatie-integratie. Hoewel het nog wel een jaartje zal duren voordat SCA breed ingezet gaat worden, zijn er enkele goede redenen om nu met SCA aan de slag te gaan. SCA helpt mee aan kostenbesparing, flexibiliteit en het verminderen van de afhankelijkheid van een ESB-product of leverancier. Kostenbesparing wordt bereikt door hergebruik van componenten en door hergebruik van de kennis van developers en beheerders. Omdat SCA-componenten een gestandaardiseerde interfacebeschrijving hebben en hun implementatie geen communicatiespecifieke code bevat, kunnen ze gemakkelijk worden ingezet in verschillende applicaties. Ook kunnen componenten daardoor eenvoudig vervangen worden door een andere implementatie.
De taal waarmee de componenten worden geassembleerd tot een composite, Service Component Description Language (SCDL), is gestandaardiseerd. Dit leidt tot hergebruik van kennis en ervaring van integratiedevelopers. Ten slotte kan een ESB die geconfigureerd wordt met SCDL eenvoudiger worden vervangen door een andere op SCDL-gebaseerde ESB, dan wanneer de ESB geconfigureerd wordt via een proprietary configuratietaal.

Volwassener

In dit artikel hebben we je een introductie willen geven in SCA. Naast een voorbeeld hebben we ook stilgestaan bij SCA-tools en het waarom van SCA. SCA heeft het afgelopen jaar een belangrijke sprong vooruit gemaakt in zijn pad naar volwassenheid door onder andere het uitkomen van de 1.0-versie van de specificatie en het overnemen van de SCA-specificatie door OASIS. De eenvoudige voorbeelden in dit artikel hebben bugreports opgeleverd voor Tuscany waaruit valt af te leiden dat Tuscany nog in de kinderschoenen staat. Omdat IBM gebruik gaat maken van Tuscany in zijn tooling, verwachten we dat dit soort problemen snel zullen worden opgelost. De komende jaren zal SCA verder ontwikkeld worden bij OASIS. SCA zal vooral groot worden in de ontwerp- en beheerdisciplines. Op het gebied van applicatiebouw is het vooral een afspraak waaraan programmeurs zich moeten houden, niet zozeer een API waartegen geprogrammeerd moet worden. We verwachten dat rond 2010 een kwart van de nieuwbouwapplicaties SCA zal gebruiken. «

Kostenbesparing wordt bereikt door hergebruik van componenten en van kennis