SOA Seminar

Service Component Architecture (SCA)

Maximilian Herold

23.01.2006

Ausarbeitung zum Seminarvortrag vom 17.01.2006
im Rahmen des Seminars zu Service-Orientierten Architekturen
an der Fachhochschule Wedel bei Prof. Dr. Sebastian Iwanowski

Inhaltsverzeichnis

1 Einleitung

Überblick *Dieses Kapitel erläutert die Motivation von SCA und gibt einen kurzen Überblick über die dahinterstehende Organisation sowie den aktuellen Status von SCA.*

1.1 Service-orientierte Architekturen

Unternehmen müssen heutzutage zunehmend in der Lage sein, schnell und flexibel auf neue Anforderungen reagieren zu können. Das können z.B. Änderungen in den Unternehmensrichtlinien sein, Regularien, Gesetze, neue Partnerschaften oder aber durch das Geschäftsumfeld bedingte Situationen (vgl. [BBB+05, S.3]). Dem gegenüber stehen häufig komplexe, gewachsene IT-Architekturen, deren Integration, Umgestaltung und Weiterentwicklung sehr schwierig ist (vgl. [KE06]). Dies ist insbesondere bedingt durch den monolithischen Aufbau der vorhandenen Applikationen und durch Nutzung von verschiedenen Kommunikationstechnologien, die an sich nicht interoperabel sind.

In einem solchen Szenario ist eine flexible Anpassung und Optimierung der Geschäftsprozesse eines Unternehmens nicht möglich. Als Beispiel sei hier das Thema Outsourcing genannt, oder auch der Wechsel eines an das System angebundenen externen Dienstleisters. Die IT sollte aber optimalerweise die Geschäftsprozesse eines Unternehmens stützen anstatt sie zu behindern.

Diese Problematik führte in den letzten Jahren zu dem Paradigma des Service-Oriented Computing bzw. als Implementierung dieses Paradigmas zu service-orientierten Architekturen (SOA) (vgl. [PTD+06, S.3]). Eine SOA besteht aus einer Kombination von verteilten Business-Funktionen[1] und -Prozessen, welche als wiederverwendbare Services mit wohldefinierten Schnittstellen und Vereinbarungen veröffentlicht sind. Services sind lose gekoppelt und unabhängig von Plattform und Implementierung. Die Umsetzung einer service-orientierten Architektur soll die Flexibilität des Gesamtsystems wesentlich erhöhen.

[1] "Business Funktion" ist hier im Gegenstatz zu "technische Funktion" zu verstehen (die Aufgabe der Funktion hat eine geschäftliche Relevanz, in Abgrenzung zu unterstützenden Aufgaben z.B. im Bereich der Infrastruktur)

1.2 Service Component Architecture

1.2.1 Hintergrund und Organisation

Bei der Umsetzung einer SOA gibt es für viele der dabei auftretenden Problemstellungen proprietäre Lösungen von verschiedenen Software-Anbietern; jedoch ist ein gemeinsames, industrieweites und einheitliches Modell bis vor kurzem nicht verfügbar gewesen. Insbesondere betrifft das

- die Entwicklung von zusammengesetzten Services sowie Netzwerken von Services und
- die Abstraktion von Middleware-Technologien.

Seit dem Jahr 2005 wird aus diesem Grund von verschiedenen interessierten Unternehmen die sog. Service Component Architecture (SCA) entwickelt. Die beteiligten Firmen sind zu diesem Zweck in der Open Service Oriented Architecture Collaboration (OSOA) organisiert. Momentan unterstützen 17 Unternehmen die Entwicklung von SCA, unter anderem BEA, IBM, Oracle, SAP, Siemens, Sun und Sybase. Ihr erklärtes Ziel ist, ein sprachunabhängiges Programmiermodell für SOA zu definieren. Dabei soll ein möglichst hohes Maß an Portabilität und Wiederverwendbarkeit der Services und Service-Implementierungen erreicht werden. SCA soll insgesamt die Erstellung und Integration von SOA-Applikationen vereinfachen. Neben SCA ist die OSOA auch noch für die sog. Service Data Objects (SDO) verantwortlich, auf die in Kapitel 6.1 (S.17) kurz eingegangen wird. Sowohl SCA als auch SDO sollen ab einem gewissen Reifegrad an eine geeignete Standardisierungsorganisation weitergereicht werden.[2]

1.2.2 Übersicht über die SCA Spezifikationen

SCA besteht aus einer Reihe von offenen Spezifikationen, in denen die Beschreibung, die Assembly[3] und das Deployment von Services beschrieben ist. Bei SCA werden diese Aspekte über Metadaten definiert, und zwar unabhängig von Implementierungssprache und Deployment-Plattform (vgl. [LBM06]). SCA umfaßt folgende Spezifikationen:

Assembly Model: ein vereinheitlichtes Modell für den Zusammenbau von sowohl lose als auch stark gekoppelten Services. Hier wird beschrieben, wie Business Funk-

[2] vgl. die offiziellen Seiten der OSOA unter http://www.osoa.org/

[3] Assembly = Aufbau, Zusammenbau Konstruktion (dieser Begriff wird anstatt einer deutschen Übersetzung im Folgenden weiterhin verwendet, da er der Name für ein zentrales SCA Konzept ist)

tionen, die aus einer oder mehreren Komponenten bestehen, zu einer zusammengesetzten Applikation verbunden werden. Es wird spezifiziert, wie die einzelnen Komponenten und deren Abhängigkeiten konfiguriert werden können. Das Assembly Model ist an verschiedenen Stellen erweiterbar[4]. Es wird im folgenden Kapitel näher vorgestellt.

Client and Implementation Models: spezifizieren die Umsetzung von SCA-Elementen für eine konkrete Implementierungssprache. Momentan sind Spezifikationen für Java, BPEL, Spring und C++ verfügbar. In Kapitel 3 (S.12) wird kurz auf das Client and Implementation Model for Java eingegangen.

Binding and Policy Framework: beschreibt die Anwendung von Infrastruktur- Funktionalitäten auf SCA Elemente zur Umsetzung von nichtfunktionalen Anforderungen (z.B. Security, Transaktionen). Die Grundkonzepte des Binding and Policy Frameworks werden in Kapitel 4 (S.14) vorgestellt.

2 Assembly Model

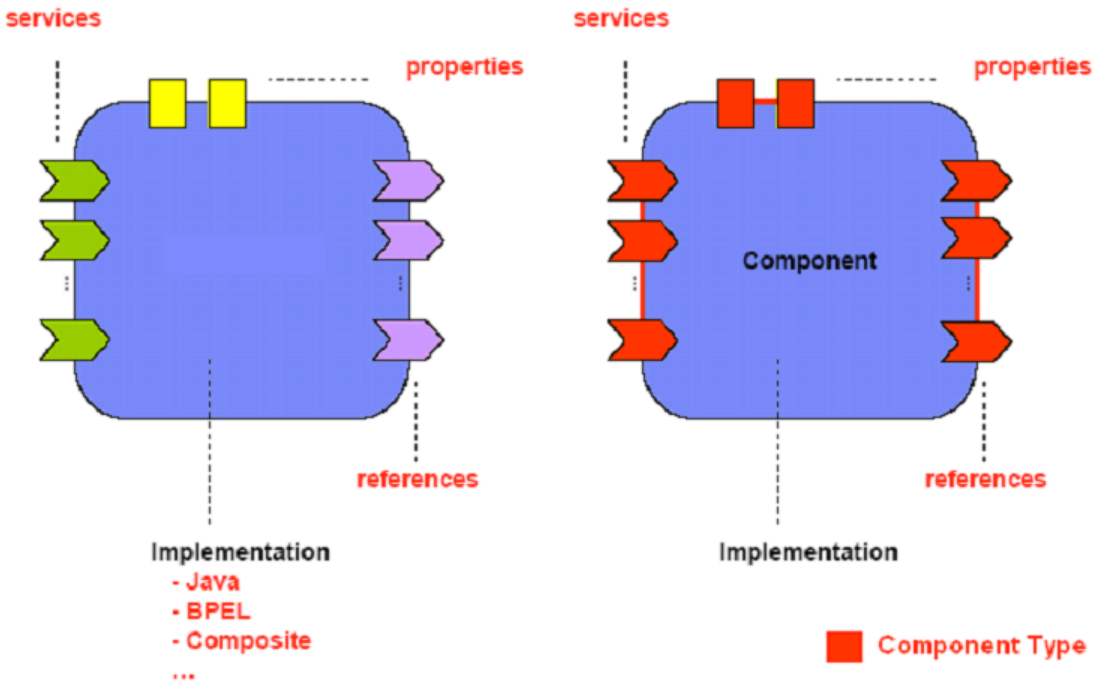
Überblick *In diesem Kapitel werden der Aufbau und die Elemente des SCA Assembly Models erläutert. Dieses bildet den Kern von SCA.*

2.1 Components

Auf unterster Ebene im SCA Modell stehen die "Components". Es soll zunächst ein Stück Programmcode in einer beliebigen Sprache betrachtet werden, der eine Business Funktion konkret umsetzt. Dieser soll nach außen hin einen oder mehrere Dienste anbieten und ist ggf. abhängig von anderen Diensten. Angebotene Dienste werden in SCA "Services" genannt, Abhängigkeiten "References" (s. Abb. 1a). Weiterhin bietet SCA die Möglichkeit, die Implementierung über "Properties" konfigurierbar zu machen. Properties sind Werte oder Parameter, die die Ausführung der Business Logik beeinflussen.

Ein Component besteht nun aus der Implementierung und den konfigurierbaren Aspekten der Implementierung ("Component Type", s. Abb. 1b): den Services, References und Properties. Dieses Konzept ermöglicht die Wiederverwendung von Implementierungen

[4] Auf die Erweiterbarkeit wird im Rahmen dieser Arbeit nicht im Detail eingegangen - der Vollständigkeit wegen sei hier nur erwähnt, dass (mit Einschränkungen) neue Interface Types, Implementation Types und Binding Types definiert werden können.



(a) (b)

Abbildung 1: Component (vgl. [OSO06b, S.3])

in verschiedenen Umgebungen mit verschiedenen Konfigurationen: Es können verschiedene Components mit jeweils denselben Implementierungen existieren. Ein Component ist also als konfigurierte Instanz einer Implementierung anzusehen.

Abbildung 2 verdeutlicht dies noch einmal: Die Implementierung (hier in Java) besitzt u.a. Properties und References, welche in Components für verschiedene Anwendungen unterschiedlich konfiguriert werden können[5]. Zur weiteren Abgrenzung wird eine Implementierung zur Laufzeit (also während der Code einer Implementierung ausgeführt wird) als "Implementation Instance" bezeichnet.

2.2 Composites, Interfaces und Bindings

Composites sind die Grundbausteine für die Komposition von Components. Ein Composite selbst verfügt ebenfalls über Properties sowie nach außen sichtbare Services und References (s. Abb. 3). Über die Properties können die Properties von in dem Composite enthaltenen Components konfiguriert werden.

[5]Der Component Type kann in diesem Zusammenhang in Relation zu einem Component betrachtet werden wie eine Klasse in Relation zu einem Objekt (nur dass ein SCA Component weit weniger dynamisch ist, da die konkrete Konfiguration sich nicht zur Laufzeit ändert).

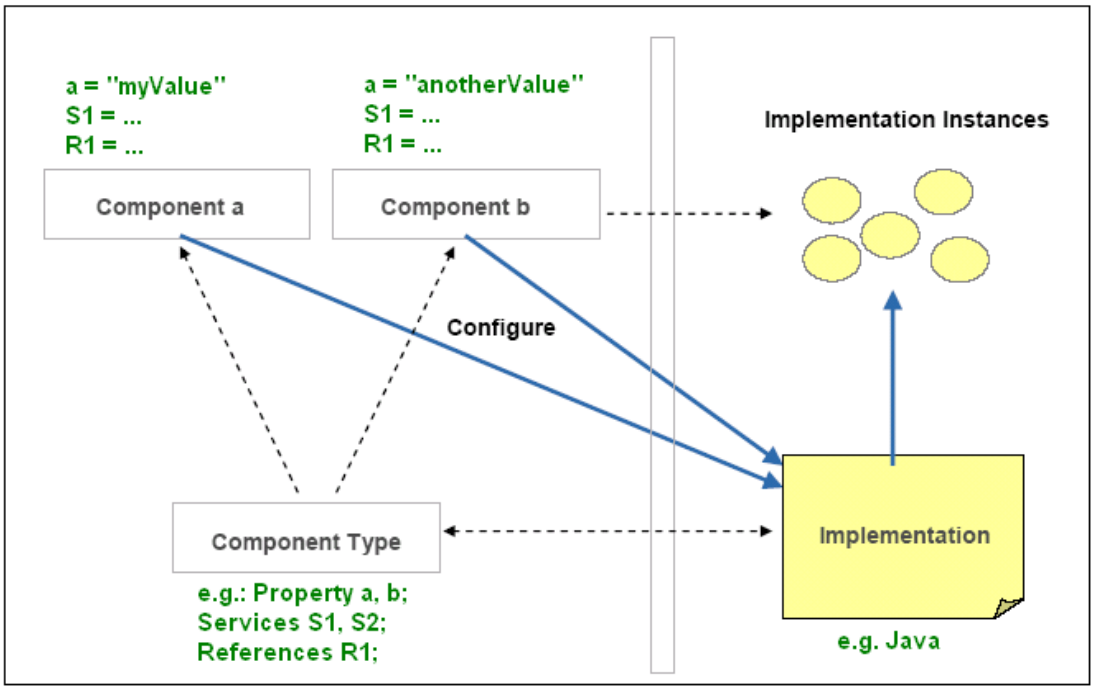


Abbildung 2: Zusammenhang zwischen und Implementation (vgl. [OSO06b, S.11])

"Wires" konfigurieren die Services und References: Zum einen wird damit festgelegt, welche Dienste und Abhängigkeiten nach extern angeboten werden bzw. von extern bedient werden müssen; zum anderen können damit Components untereinander verknüpft werden. Im Beispiel aus Abb. 3 wird also die Implementierung von Component A zur Laufzeit den angebotenen Service von Component B nutzen.

Die Schnittstelle von Services nach außen werden über "Interfaces" beschrieben. Jeder Service und jede Reference muss demnach ein Interface besitzen. Für die Schnittstellenbeschreibung können in SCA momentan Java Interfaces, WSDL 1.1 Port Types oder WSDL 2.0 Interfaces genutzt werden[6].

Neben der Beschreibung der Schnittstelle ist für das Aufrufen und Bereitstellen von Services noch die Definition konkreter Kommunikationstechnologien nötig. Dies geschieht in SCA über "Bindings", die ebenfalls für jeden Service und jede Reference definiert werden müssen. Momentan spezifiziert SCA konkrete Bindings für Web Services, JCA, JMS, stateless session EJBs sowie einem internen SCA Format[7].

[6] D.h. auch eine Implementierung in z.B. Ruby, JavaScript oder XSL benötigt für die Beschreibung der Interfaces momentan eine der drei genannten Technologien.

[7] Das SCA Binding ist für effiziente Kommunikation innerhalb eines SCA Systems eines Herstellers gedacht; es wird ausdrücklich keine Interoperabilität von diesem Binding verlangt.

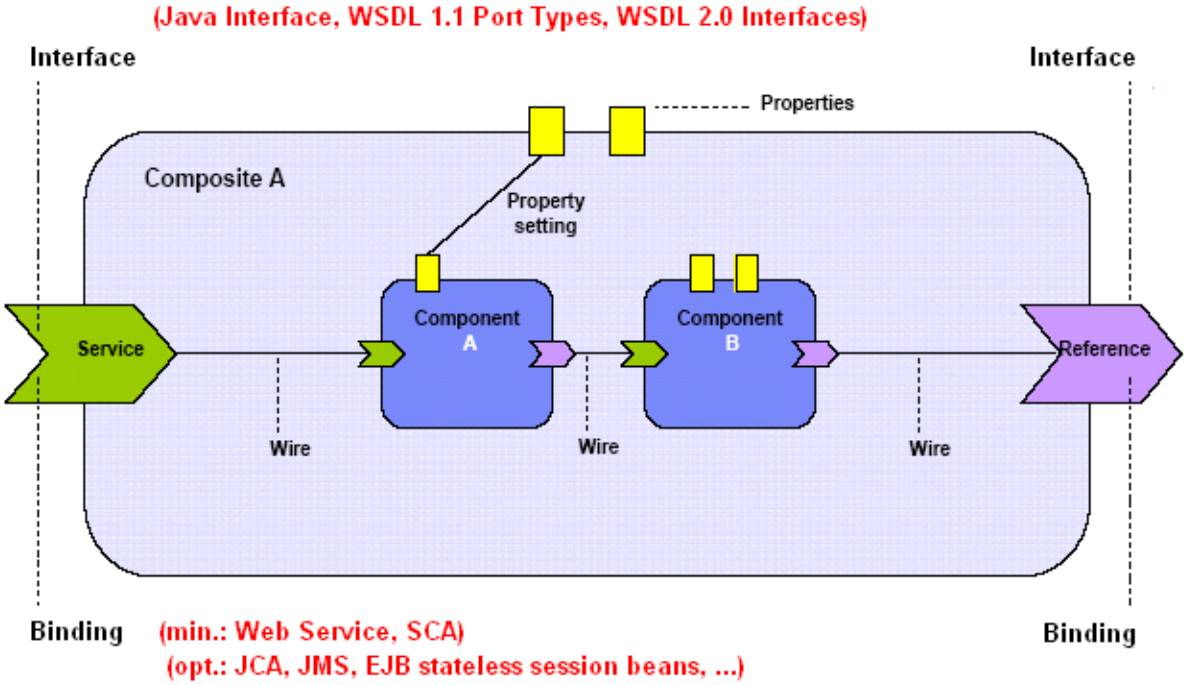


Abbildung 3: Composite (vgl. [KE06])

2.3 Beispiel: Composites als Components

Die Umsetzung des Assembly Models über Metadaten soll hier an einem kleinen Beispiel erklärt werden. Dabei wird auch die Rekursion im Assembly Model deutlich: Neben z.B. Java Code können auch andere SCA Composites als Implementierung für Components dienen (s.a. Abb. 1a auf Seite 6). Das ermöglicht eine beliebig tiefe Schachtelung von Composites und Components und damit die Möglichkeit, feingranularere Funktionalitäten flexibel und modular zu grobgranulareren Services zusammenzustellen.

In dem Beispiel soll ein *OrderFulfillmentComposite* einen Service in einem fiktiven Unternehmen bereitstellen, der nach einem Bestellvorgang das Besorgen der bestellten Produkte erledigt. Dabei gebe es hier zwei Arten von Produkten: Eine (Produktart A) sei nicht im Unternehmen verfügbar und muss deshalb von extern bezogen werden; die andere (Produktart B) sei immer intern auf Lager. Für die Bestellung im Lager gebe es bereits einen *StockComposite* (s. Abb. 4). Weiterhin sei ein *OrderFulfillmentServiceComposite* vorhanden, der eine Bestellung entgegennimmt und für jedes bestellte Produkt je nach Produktart (A oder B) die Bestellung an einen anderen Service delegiert. Die vorhandenen Elemente sollen nun in einem *OrderFulfillmentComposite* zusammengebaut und konfiguriert werden.

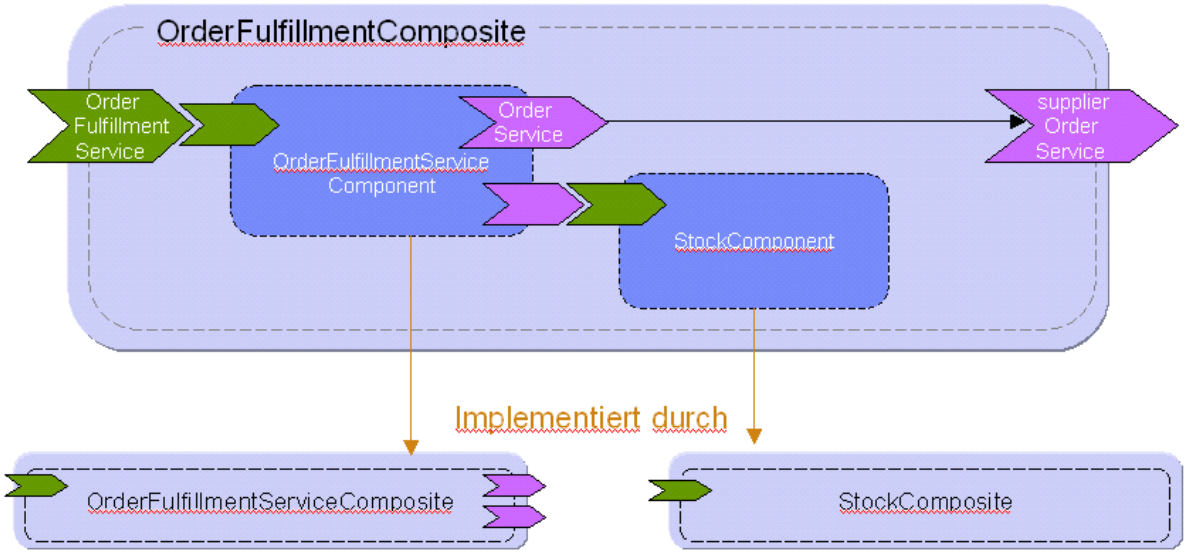


Abbildung 4: Beispiel: OrderFulfillmentComposite

Dazu werden zunächst zwei Components definiert (*OrderFulfillmentServiceComponent* und *StockComponent*), die als Implementierung den jeweiligen vorhandenen Composite nutzen (s. Listing 1 auf der folgenden Seite, Zeilen 19-27). Für den *OrderFulfillmentComposite* wird nun ein Service für die Entgegennahme der Bestellung definiert (*OrderFulfillmentService*, siehe Zeilen 6-11). Als Interface-Beschreibung wird hier ein Java Interface verwendet. Weiterhin soll der Service über einen Web Service erreichbar sein, deshalb in Zeile 8f. die Angabe eines Web Service Bindings[8]. Wenn der Service mit einer Bestellung aufgerufen wird, soll der Aufruf an den *OrderFulfillmentServiceComponent* weitergereicht werden; Zeile 10 beschreibt das entsprechende Wire[9].

Die interne Bestellung von Produktart B wird vom *OrderFulfillmentServiceComponent* an den *StockComponent* weitergeleitet (Z.21); die externe Bestellung von Produktart A an die Reference des *OrderFulfillmentComposites* (Z.22).

[8] andere Technologien erfordern hier die Angabe anderer Paremeter (z.B. bei einem RMI-Binding: Host, Port und Name des Services)

[9] Wires können alternativ auch über ein `<wire/>` Element in der darüberliegenden Ebene mit Angabe von Quelle und Ziel spezifiziert werden.

```
<?xml version="1.0" encoding="UTF-8"?>
<composite xmlns:xsd="http://www.w3.org/2001/XMLSchema-instance"
          xmlns="http://www.osoa.org/xmlns/sca/1.0"
          name="OrderFulfillmentComposite">

<service name="OrderFulfillmentService">
   <interface.java interface="services.order.OrderFulfillmentService"/>
   <binding.ws port="OrderFulfillmentService#
  wsdl.endpoint(OrderFulfillmentService/OrderFulfillmentServiceSOAP)"/>
   <reference>OrderFulfillmentServiceComponent</reference>
</service>

<reference name="supplierOrderService">
   <interface.java interface="services.supplier.order.OrderService"/>
   <binding.ws port="http://www.mysupplier.com/OrderService#
             wsdl.endpoint(OrderService/OrderServiceSOAP)"/>
</reference>

<component name="OrderFulfillmentServiceComponent">
   <implementation.composite name="OrderFulfillmentServiceComposite"/>
   <reference name="StockService">StockComponent</reference>
   <reference name="OrderService">supplierOrderService</reference>
</component>

<component name="StockComponent">
   <implementation.composite name="StockComposite"/>
</component>

</composite>
```

Listing 1: Order Fulfillment Composite Metadaten

2.4 SCA System und Deployment

Die Definition von Composites kann auf mehrere Dateien aufgeteilt werden: die einzelnen Teile werden dann über `<include name="..."/>` textuell eingebunden. Diese Möglichkeit wird von SCA selbst beim Deployment in eine SCA Laufzeitumgebung genutzt. Ein SCA System verhält sich ähnlich wie ein Composite: über Includes werden die deployten SCA-Elemente integriert. In einem SCA System können ausschließlich

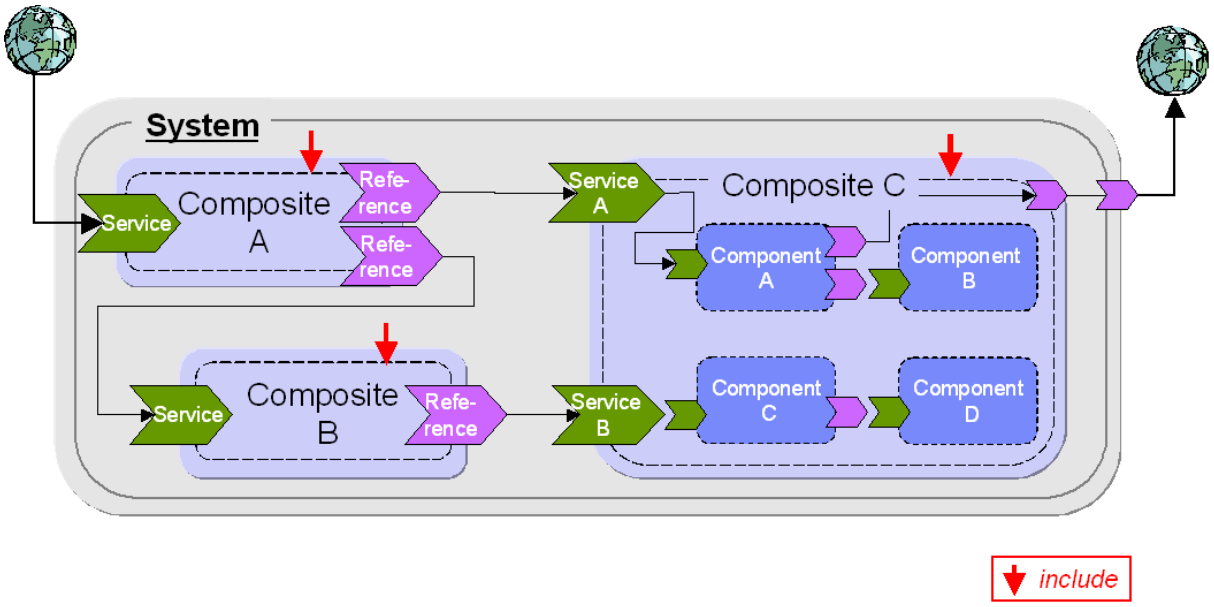


Abbildung 5: SCA System (vgl. [OSO06c])

Composites eingebunden werden (s. Abb. 5).

Die Top-Level Composites dürfen teilweise unvollständig sein. Beim Deployment können dann die fehlenden Angaben gemacht werden. Das ist insbesondere hilfreich für die Wires im SCA System und die konkreten Bindings (also wie die Services von außen erreichbar sein sollen bzw. welche externe Services genutzt werden sollen).

2.5 Klassifizierungen von Interfaces

2.5.1 Remotable und Local Interfaces

Wie schon am Beispiel der möglichen Bindings deutlich wurde (vgl. Kap. 2.2), zielt SCA auch auf die Integration interner, stark gekoppelter Services ab. Um eine möglichst hohe Performanz zu ermöglichen, gibt es auf der Ebene der Interfaces deshalb die Möglichkeit, eine Schnittstelle so zu definieren, dass ein Datenaustausch by-reference möglich ist. Diese werden als “Local” Interfaces bezeichnet. Im Gegensatz dazu erlauben die sog. “Remotable” Interfaces nur by-value-Datenaustausch. Außerdem ist ein Überladen von Methoden hier verboten. Dafür können Services mit Remotable Interfaces von Clients aufgerufen werden, die auf einem anderen Betriebssystem laufen. (vgl. [OSO06b, S.14f.])

2.5.2 Bidirectional und Conversational Interfaces

Bisher wurde ausschließlich die synchrone Kommunikation zwischen Services betrachtet. SCA sieht allerdings auch Möglichkeiten zur Definition von asynchroner Kommunikation vor. Für bidirektionale Peer-to-Peer Verbindungen gibt es dafür die sog. "Bidirectional" Interfaces. Zu einem Service wird dafür ein zusätzliches Callback-Interface in den Metadaten definiert, über welches ein aufgerufener Service zu einem späteren Zeitpunkt seine Antwort schicken kann. (vgl. [OSO06b, S.15])

Weiterhin gibt es Services, die nur durch eine Folge zusammenhängender Operationen beschrieben werden können. Dafür muss über mehrere Aufrufe hinweg ein Zustand gehalten werden. Auf Applikations-Ebene könnte man dieses Problem durch die Übergabe eines zusätzlichen Parameters (z.B. eine Konversations-ID) lösen. Das ist ebenso mit wie ohne SCA möglich; SCA kann jedoch alternativ mit den sog. "Conversational" Interfaces sowohl das Zuordnen der Nachrichten als auch das Lebenszyklus-Management des Services unterstützen. Voraussetzung für Conversational Interfaces ist allerdings, dass das genutzte Binding diese Funktionalität unterstützt[10]. (vgl. [OSO06b, S.15ff.])

3 Client and Implementation Model for Java

Überblick *Dieses Kapitel geht exemplarisch auf einige Details des SCA Client and Implementation Models for Java ein, um die Umsetzung von SCA in einer bestimmten Implementierungssprache zu beleuchten.*

Das Client and Implementation Model for Java spezifiziert die Abbildung von SCA Elementen und Konzepten auf Java. Dies soll hier an zwei kleinen Beispielen demonstriert werden.

In Beispiel 1 geht es um die Implementierung eines Services in Java (HelloService, s. Abb. 6b), der auch von außerhalb des Betriebssystems aufrufbar sein soll. Im dazugehörigen Listing 2 ist das Java Interface, welches als SCA Remotalbe Interface für den HelloService dient, deshalb mit der Annotation `@Remotable` versehen. Die Implementierung selbst ist mit `@Service(HelloService.class)` markiert, um dem SCA Laufzeitsystem mitzuteilen, welche von `HelloService` erbende Klasse bei einem Aufruf des Services genutzt werden sollen.

[10] z.B. Web Service Binding mit WS-RM oder WS-Addressing

(a) ClientService (b) HelloService

Abbildung 6: Beispiel: Components mit ClientService und HelloService

```
1  package services.hello;
2  import org.osoa.sca.annotations.*;
3
4  @Remotable
5  public interface HelloService {
6    String hello(String message);
7  }
8
9  ------------------------------------------------------
10
11 package services.hello;
12 import org.osoa.sca.annotations.*;
13
14 @Service(HelloService.class)
15 public class HelloServiceImpl implements HelloService {
16   public String hello(String message) {
17     ...
18   }
19 }
```

Listing 2:]HelloService Implementierung in Java [OSOA06a, S.2f.]

In Beispiel 2 geht es um einen anderen Component, der einen ClientService zur Verfügung stellt und von einem HelloService abhängig ist (s. Abb. 6a). Die Implementierung wird zur Laufzeit von dem SCA Laufzeitsystem mit dem konkreten HelloService, den der Client nutzen soll, konfiguriert[11]. Dieser konkrete HelloService wird dann zur Laufzeit (entsprechend der Binding und Wiring Konfiguration) z.B. per Web Service oder RMI die Methoden eines entfernten HelloService aufrufen. Die für das SCA Laufzeitsystem zu konfigurierende Referenz wird im Java Code mit der Annotation `@Reference(...)`

[11] Diese Verfahrensweise nennt sich “Dependency Injection” und ist insbesondere aus dem Spring Framework bekannt.

versehen (s. Listing 3, Zeile 10).

```
1  package services.client;
2  import services.hello.HelloService;
3  import org.osoa.sca.annotations.*;
4
5  @Service(ClientService.class)
6  public class ClientServiceImpl implements ClientService {
7
8    private HelloService helloService;
9
10   @Reference(name="helloService", required=true)
11   public void setHelloService(HelloService service){
12     helloService = service;
13   }
14   public String clientMethod() {
15     String result = helloService.hello("Hello World!");
16     ...
17   }
18 }
```

Listing 3:]ClientService Implementierung in Java [OSOA06a, S.3]

Neben der eben beispielhaft gezeigten Umsetzung der SCA Elemente definiert die Spezifikation auch eine SCA API für Java. Damit ist es möglich, in einer Java Implementierung direkt auf SCA Objekte zuzugreifen, was in verschiedenen Situationen sinnvoll oder nötig sein kann (z.B. beim Testen).

4 Binding and Policy Framework

Überblick *In diesem Kapitel wird das Konzept von SCA vorgestellt, nichtfunktionale Anforderungen und Infrastruktur-Fähigkeiten zu beschreiben.*[12]

4.1 Übersicht

Das SCA Binding and Policy Framework spezifiziert, wie Anforderungen und Fähigkeiten nichtfunktionaler Art in einem SCA System spezifiziert werden. Das umfaßt

[12] Hinweis: Die in Kapitel 4.2 vorgestellten Bausteine der Spezifikation sind nicht vollständig, sie sollen nur das Prinzip verdeutlichen - eine detaillierte Betrachtung liegt außerhalb des Rahmens dieser Arbeit.

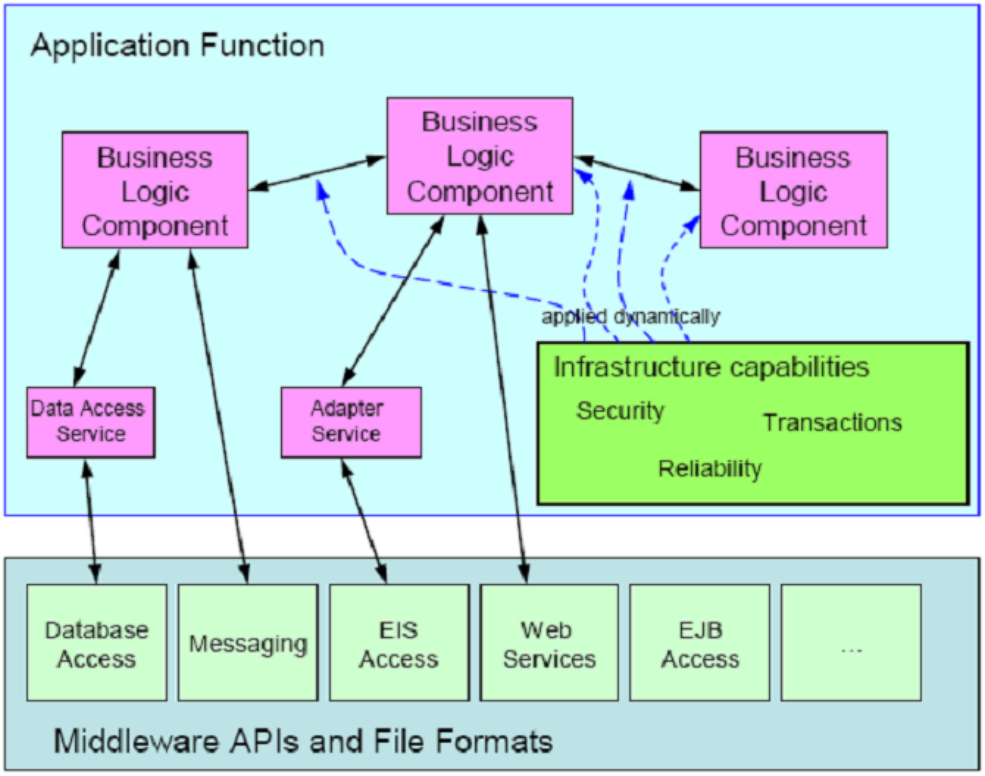


Abbildung 7: SCA Architektur - Integration von Policies [BBB+05, S.5]

Fähigkeiten der Infrastruktur, u.a. Sicherheit, Zuverlässigkeit und Transaktionen. Der initiale Fokus der OSOA liegt auf diesen genannten Bereichen.

Im Prinzip funktioniert die Konfiguration der nichtfunktionalen Aspekte ähnlich wie die Konfiguration im Assembly Model: Policies werden über Metadaten an SCA Elemente angehängt (s. Kasten "Infrastructure Capabilities" in Abb. 7). Hier läßt sich noch unterscheiden, ob sich die Policies auf die Kommunikation zwischen Service Anbieter und Client beziehen (wie z.B. bei Vertraulichkeit) - dann werden sie "Interaction Policies" genannt und auf Services und References angewendet. Beziehen sich die Policies hingegen auf Fähigkeiten, die der Container bereitstellen soll, in dem ein Component läuft (z.B. Transaktionen), werden sie "Implementation Policies" genannt und auf Components angewendet. (vgl. [OSO06a])

4.2 Grundbausteine

Wie auch im Assembly Model, kann beim Binding and Policy Framework zunächst eine abstrakte Beschreibung vorgenommen werden, die später dann an eine konkrete Technologie und Umsetzung gebunden wird. Abstrakte, high-level Anforderungen werden

über "Intents" spezifiziert. Diese legen (fast) nur namentlich z.B: ‘‘`confidentiality`’’ oder ‘‘`confidentiality/high`’’ fest, ohne Angabe einer konkreten Implementierung oder Technologie dafür (siehe Listing 4, Zeile 3).

Über "Policy Sets" wird die Abbildung von konkreten Policies auf Intents festgelegt. Die Angabe von Policy Sets erfolgt wahlweise beim Deployment oder direkt in einem Binding- bzw. Component-Element (siehe Listing 5, Zeile 3). Konkrete Policies werden in SCA momentan über WS-Policy und WS-PolicyAttachment definiert.

```
1 <sca:service name="mySpecialService">
2   <sca:interface.wsdl portType="..."/>
3   <sca:profile intents="sec.authentication rel.reliabilty"/>
4 </sca:service>
```

Listing 4: Beispiel: Integration von Intents [BKM+06]

```
1 <sca:reference name="FlightService">
2     <sca:interface ... />
3     <sca:binding.WS policySet="BasicSecurity"/>
4 </sca:reference>
```

Listing 5: Beispiel: Integration von Policy Sets [BKM+06]

5 Werkzeuge

Überblick *Dieses Kapitel gibt eine kurze Übersicht von Werkzeugen im Umfeld von SCA.*

Es gibt bereits diverse kommerzielle Werkzeuge, die SCA implementieren, u.a. von IBM (WebSphere Application Server) und Oracle (EDA Suite)[13]. Keines der kommerziellen Produkte wurde allerdings im Rahmen dieser Arbeit näher betrachtet.

Im Open Source Bereich gibt es ebenfalls SCA Implementierungen. Das SOA PHP Projekt stellt eine SCA Laufzeitumgebung für PHP zur Verfügung (PHP PECL SCA/SDO[14]). Das Eclipse SOA Tools Platform Project (STP)[15], in welchem diverse Tools im

[13] s.a.: http://www.osoa.org/display/Main/Early+Implementation+Examples+and+Tools
[14] Homepage: http://pecl.php.net/package/SCA_SDO
[15] Homepage: http://www.eclipse.org/stp/

SOA-Umfeld entwickelt werden sollen, implementiert in seinem "Core" Subprojekt das SCA Assembly Model.

Für Java und C++ entwickelt das Apache Tuscany Projekt eine SCA Laufzeitumgebung[16]. Momentan hat das Projekt "Incubator" Status bei Apache. Es stellt auch Implementierungen für Service Data Objects (SDO) und Data Access Services (DAS) bereit[17]. Leider waren bei der Evaluierung von Tuscany durch den Autor einige benötigte Repositories nicht verfügbar, so dass nicht alle Funktionalitäten getestet werden konnten. Es entstand allerdings der Eindruck, dass die SCA Funktionalitäten prinzipiell alle vorhanden sind. Wie auch der Projektstatus suggeriert, rät der Autor von einem produktiven Einsatz zu diesem Zeitpunkt ab, sieht jedoch einiges an Potential für Tuscany in der Zukunft - insbesondere wenn die Benutzerfreundlichkeit erhöht werden kann sowie graphische SCA-Editoren verfügbar sind.

6 Verwandte Technologien

Überblick *.Hier wird auf die in dieser Arbeit oder in dem dazugehörigen Vortrag angesprochenen Technologien eingegangen, die SCA ähnlich sind oder in einer anderen Form mit SCA zusammenhängen.*

6.1 Service Data Objects und Data Access Services

Während bei SCA ein einheitlicher Zugriff auf Services thematisiert wird, beschäftigen sich Service Data Objects (SDO) mit einem einheitlichen Verarbeitungsmodell für ausgetauschte Daten. SDOs werden ebenfalls von der OSOA spezifiziert und sind schon wesentlich ausgereifter als SCA (momentan Version 2.x). In den SCA Spezifikationen wird SDO als bevorzugtes Modell für ausgetauschte Daten behandelt.

SDOs wurden aufgrund der fehlenden allgemeinen Verfügbarkeit von Technologien wie JAXB und JDO entwickelt. Sie bieten flexible Datenstrukturen und einen einheitlichen Zugriff auf Daten aus verschiedenen Quellen. Unter anderem wird auch der Zugriff auf Daten geregelt, die "detached" sind (die also zwischenzeitlich keine Verbindung zu der ursprünglichen Datenquelle haben). SDO zeichnet sich außerdem durch eine gute XML- und XML Schema-Integration aus. Die sogenannten Data Access Services (DAS)

[16] Homepage: http://incubator.apache.org/tuscany/
[17] siehe Kapitel 6.1

sind verantwortlich für das Bewegen von Daten zwischen Datenquellen und SDOs. (vgl. [GW06])

6.2 Windows Communication Foundation

Seit dem Jahre 2003 wurde von Microsoft die Windows Communication Foundation entwickelt (WCF, Codename "Indigo"). WCF hat ähnliche Ziele wie SCA und damit auch einige Gemeinsamkeiten. Als Beispiele seien hier genannt: die Möglichkeit, aus "normalen" Objekten service-orientierte Applikationen konstruieren zu können, das Konzept von Bindings, sowie die Nutzung von WS-* Spezifikationen. WCF ist allerdings rein .NET-basiert und spezifiziert keine Assembly von Komponenten und keine Konfiguration über Wires. Dafür focussiert WCF aber nicht auf eine spezielle Technologie für den Datenaustausch (SCA präferiert SDO, siehe vorangeganener Abschnitt). (vgl. [Cha05])

7 Fazit

Überblick *Dieses Kapitel fasst die wichtigsten Aspekte zusammen.*

SCA soll ein einheitliches industrieweites Programmiermodell für service-basierte Applikationen werden, welches deren Erstellung vereinfacht sowie die Wiederverwendbarkeit und Integrationsfähigkeit erhöht. Angesichts der Unterstützung von SCA durch die meisten der wichtigen Unternehmen in diesem Bereich (bis auf Microsoft) ist zu erwarten, dass sich SCA konsequent in diese Richtung weiterentwickeln und mit großer Wahrscheinlichkeit diesem Ziel gerecht werden wird.

Die Kernideen sollen hier zum Schluß noch einmal herausgehoben werden:

- die Modulare Komposition von Komponenten, die in verschiedenen Implementierungssprachen umgesetzt sein können;
- die Trennung der Business Logik von Middelware-Abhängigkeiten und nichtfuktionalen Aspekten ("Separation of Concerns"), wobei konkrete Technologie-Entscheidungen durch Bindings getroffen werden;
- die Wiederverwendbarkeit von Implementierungen durch Anbringen unterschiedlicher Konfigurationen.

Abbildungsverzeichnis

Akronyme

DAS	Data Access Services
EJB	Enterprise Java Bean
JAXB	Java Architecture for XML Binding
JCA	Java Connector Architecture
JDO	Java Data Objects
JMS	Java Messaging System
OSOA	Open Service Oriented Architecture (Collaboration)
RMI	Remote Method Invocation
SCA	Service Component Architecture
SDO	Service Data Objects
SOA	Service-orientierte Architektur
WCF	Windows Communication Foundation
WS-RM	WS-Reliable Messaging
WSDL	Web Service Description Language

Literatur

[BBB+05] Beisiegel, Michael ; Blohm, Henning ; Booz, Dave ; Dubray, Jean-Jacques ; Colyer, Adrian ; Edwards, Mike ; Ferguson, Don ; Flood, Bill ; Greenberg, Mike ; Kearns, Dan ; Marino, Jim ; Mischkinsky, Jeff ; Nally, Martin ; Pavlik, Greg ; Rowley, Mike ; Tam, Ken ; Trieloff, Carl: Service Component Architecture - Building Systems using a Service Oriented Architecture / BEA, IBM, Interface21, IONA, Oracle, SAP, Siebel, Sybase. Version: November 2005. http://www.iona.com/devcenter/sca/SCA_White_Paper1_09.pdf. – Whitepaper. – Online–Ressource, Abruf: 2007-01-16

[BKM+06] Beisiegel, Michael ; Kavantzas, Nickolas ; Malhotra, Ashok ; Pavlik, Greg ; Sharp, Chris: SCA Policy Association Framework. In: Dan, Asit (Hrsg.) ; Lamersdorf, Winfried (Hrsg.): *Service-Oriented Computing - ICSOC 2006, 4th International Conference* Bd. 4294 (Lecture Notes in Computer Science), Springer, 2006, S. 613–623

[Cha05] Chappell, David: *Foundations for Service-Oriented Applications - Comparing WCF and SCA*. Blog. Version: November 2005. http://www.davidchappell.com/HTML_email/Opinari_No15_12_05.html. – Online-Ressource, Abruf: 2007-01-22

[GW06] Goodson, Kelvin ; Winn, Geoffrey: SOA & Web Services - What Is SDO? - Part One: The value of many of the facets of SDO. In: *Java Developer's Journal* (2006), Dezember. http://java.sys-con.com/read/313547.htm. – Online-Artikel, Abruf: 2007-01-11

[KE06] Karmarkar, Anish ; Edwards, Mike: Assembly of Business Systems using Service Component Architecture. In: Dan, Asit (Hrsg.) ; Lamersdorf, Winfried (Hrsg.): *Service-Oriented Computing - ICSOC 2006, 4th International Conference, Chicago, IL, USA* Bd. 4294 (Lecture Notes in Computer Science), Springer, 529-539

[LBM06] Laws, Simon ; Borley, Andrew ; Mahbod, Haleh: Real SOA - Web Services and Service Oriented Architecture - An overview of SCA and SDO.

In: *Java Developer's Journal* (2006), Dezember. http://java.sys-con.com/read/299972.htm. – Online Artikel, Abruf: 2007-01-11

[OSO06a] OSOA: *SCA Policy Framework*. Version: November 2006. http://www.osoa.org/download/attachments/35/SCA_Policy_Framework_v0502.pdf?version=1. – Specification, Abruf: 2007-01-10

[OSO06b] OSOA: *SCA Service Component Architecture Assembly Model Specification*. Version: August 2006. http://www.osoa.org/download/attachments/35/SCA_AssemblyModel_V096.pdf?version=1. – Specification, Abruf: 2007-01-06

[OSO06c] OSOA: *Service Component Architecture - Technology Overview*. Version: 2006. http://www.osoa.org/download/attachments/250/SCA+Technology+Overview+-+Short.pdf?version=2. – Online Ressource, Abruf: 2007-01-10

[PTD+06] Papazoglou, Michael P. ; Traverso, Paolo ; Dustdar, Schahram ; Leymann, Frank ; Krämer, Bernd J.: Service-Oriented Computing: A Research Roadmap. In: Cubera, Francisco (Hrsg.) ; Krämer, Bernd J. (Hrsg.) ; Papazoglou, Michael P. (Hrsg.): *Service Oriented Computing (SOC)*, Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany (Dagstuhl Seminar Proceedings 05462). – ISSN 1862–4405