SOA Seminar WS2006 (bei Prof. Dr. Sebastian Iwanowski)

fhwedel
UNIVERSITY OF APPLIED SCIENCES

Service Component Architecture (SCA)

Maximilian Herold (ms8329)

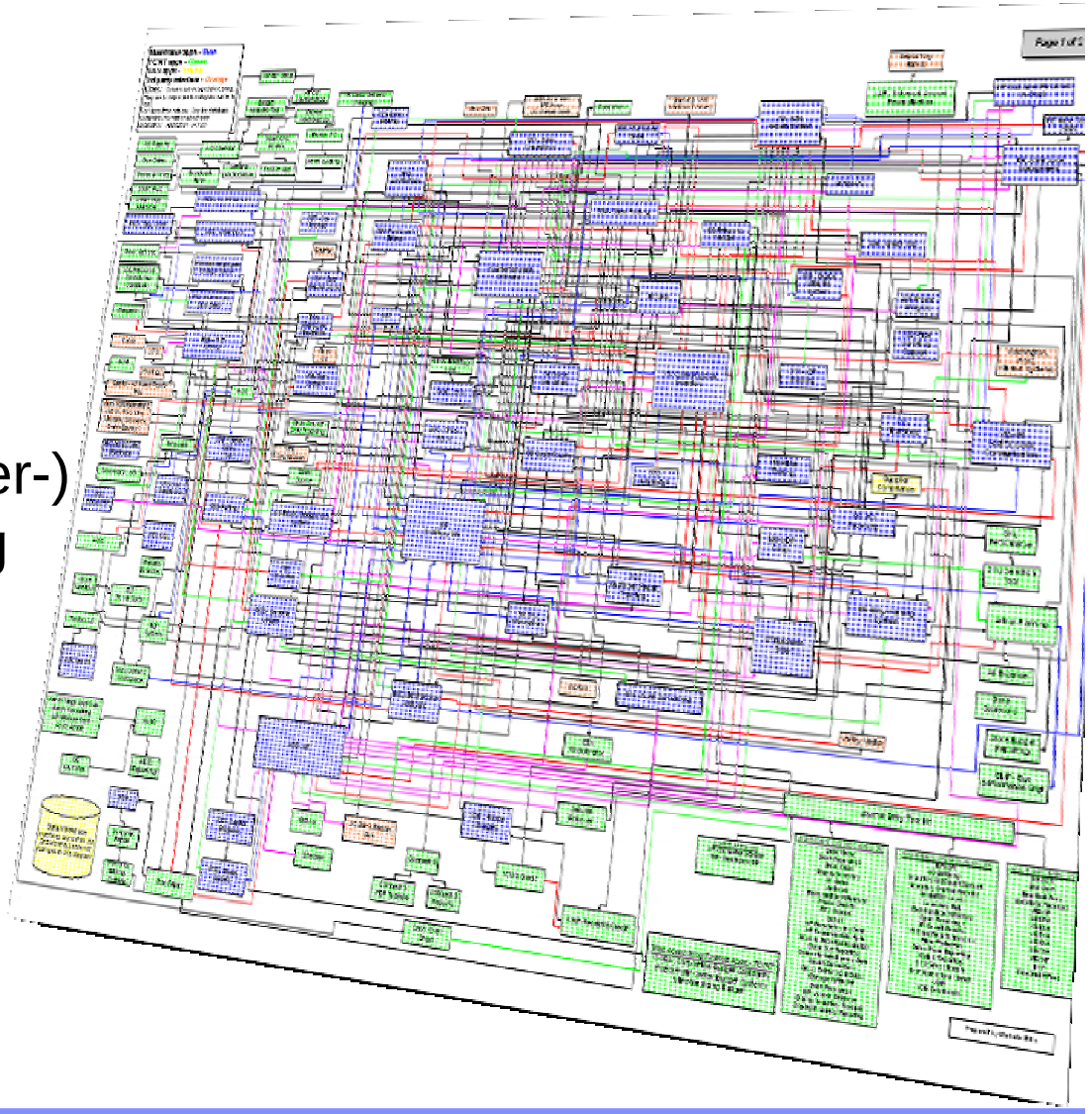
17.01.2007

Übersicht

- **Einleitung: Von SOA zu SCA**
- **SCA Spezifikationen (Teil 1)**
  - Überblick
  - Assembly Model
  - Client and Implementation Model for Java
- **Tools und Live Demo**
- **SCA Spezifikationen (Teil 2)**
  - Binding & Policy Framework
- **Fazit**

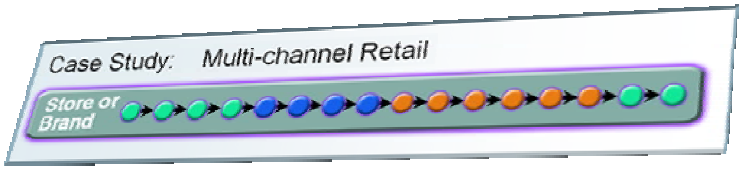
Motivation für SOA

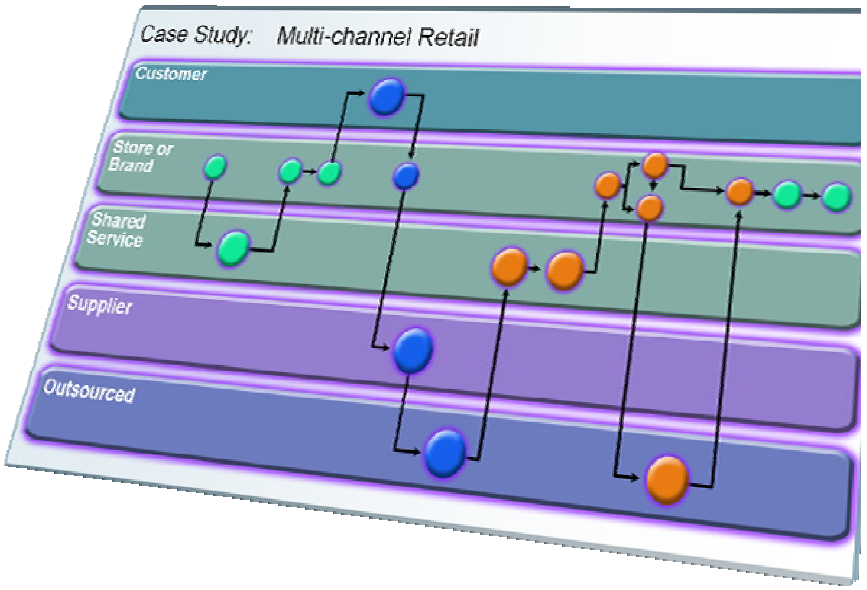
- **Probleme heute**
  - Komplexität
  - Starre Architekturen
  - Integration und (Weiter-) Entwicklung schwierig
  - Aber: Schnellebige Business-Anforderungen sowie technologische Entwicklungen



Motivation für SOA

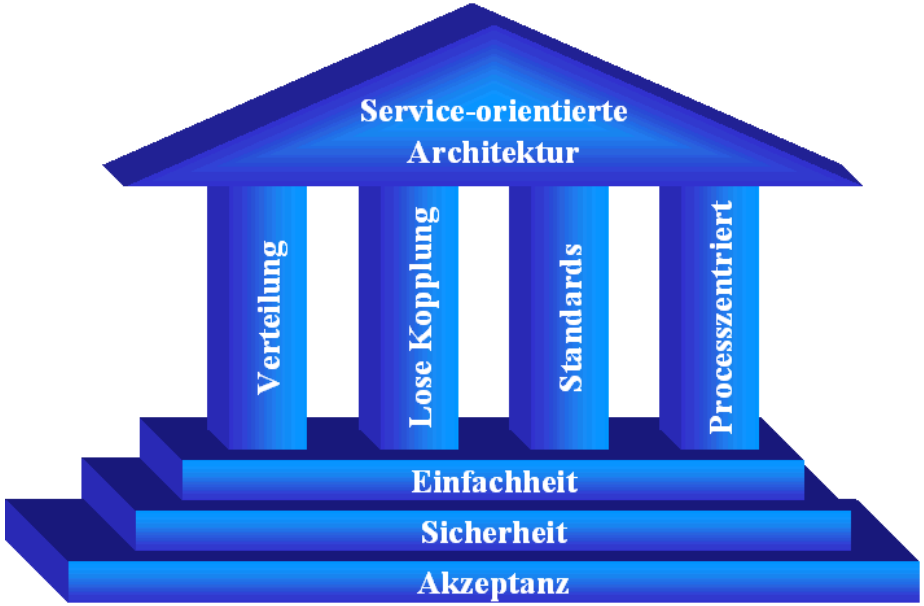
- **SOA soll ermöglichen:**
  - Schnelle Aktion und Reaktion auf neue Anforderungen
  - Flexible Anpassung und Optimierung der Geschäftsprozesse
  - Geschäftsprozess-getriebene Integration vorhandener Applikationen
  - B2B Integration





SOA

- **Charakteristika**
  - Kombination von (verteilten) Business Funktionen und Prozessen
  - als wiederverwendbare Services veröffentlicht
  - lose gekoppelt
  - Unabhängigkeit von Plattform und Implementierung
  - wohldefinierte Schnittstellen und Vereinbarungen



- **Frage: Wie realisiert man eine SOA?**

Beispiel: Ein neuer Service (ohne SCA)

- **Vorgehen: Beispiel mit Java und Apache Axis**
  - Erstellung von WSDL
  - Erstellung von Java-Klassen mit Business Logik sowie Funktionalität für die Verarbeitung der SOAP-Nachrichten
  - Deployment auf Tomcat mit Axis Runtime
- **Denkbare Problemstellungen**
  - Anbindung zusätzlich zu Web Service auch via JMS
  - Entwicklung von neuen zusammengesetzten Services sowie Netzwerken von Services
  - Einbindung von in BPEL, C++, ... implementierten Funktionalitäten

Fragen zur Realisierung einer SOA

- **Wie kann man...**
  - Business Komponenten entwickeln, die jeweils eine bestimmte Funktionalität (Services) zur Verfügung stellen?
  - zusammengesetzte Services umsetzen?
  - bei der Implementierung Abhängigkeiten zur Middleware vermeiden (→ komplex!) und damit auch die Business-Logik vor Technologie-Änderungen schützen?
  - ein hohes Maß an Portabilität und Wiederverwendbarkeit erreichen?

Entstehung von SCA

- **Open SOA Collaboration (www.osoa.org)**
  - Informelle Gruppe von Unternehmen
    - BEA, IBM, Oracle, RedHat, SAP, Siemens, Software AG, Sun, Sybase, ...
  - Ziel: Definition eines sprachunabhängigen Programmiermodells für SOA
  - Projekte: SCA und SDO (Service Data Objects)
    - Stand SCA Spezifikationen
      - Nov 2005: V0.9; aktuell V0.95 / V0.96
    - Geplant: spätere Übermittlung an eine geeignete Standardisierungs-Organisation

Was ist SCA?

- **Programmiermodell für service-basierte Lösungen**
- **Ziel: Erstellung und Integration von SOA-Applikationen vereinfachen**
- **Offene Spezifikation für**
  - Beschreibung,
  - Assembly und
  - Deployment von Services
- **Eigenschaften**
  - Unabhängig von Implementierungs-Sprache und Deployment-Plattform
  - Business-Logik getrennt von Middleware-Abhängigkeiten / APIs
- **Laufzeitumgebung für SCA**

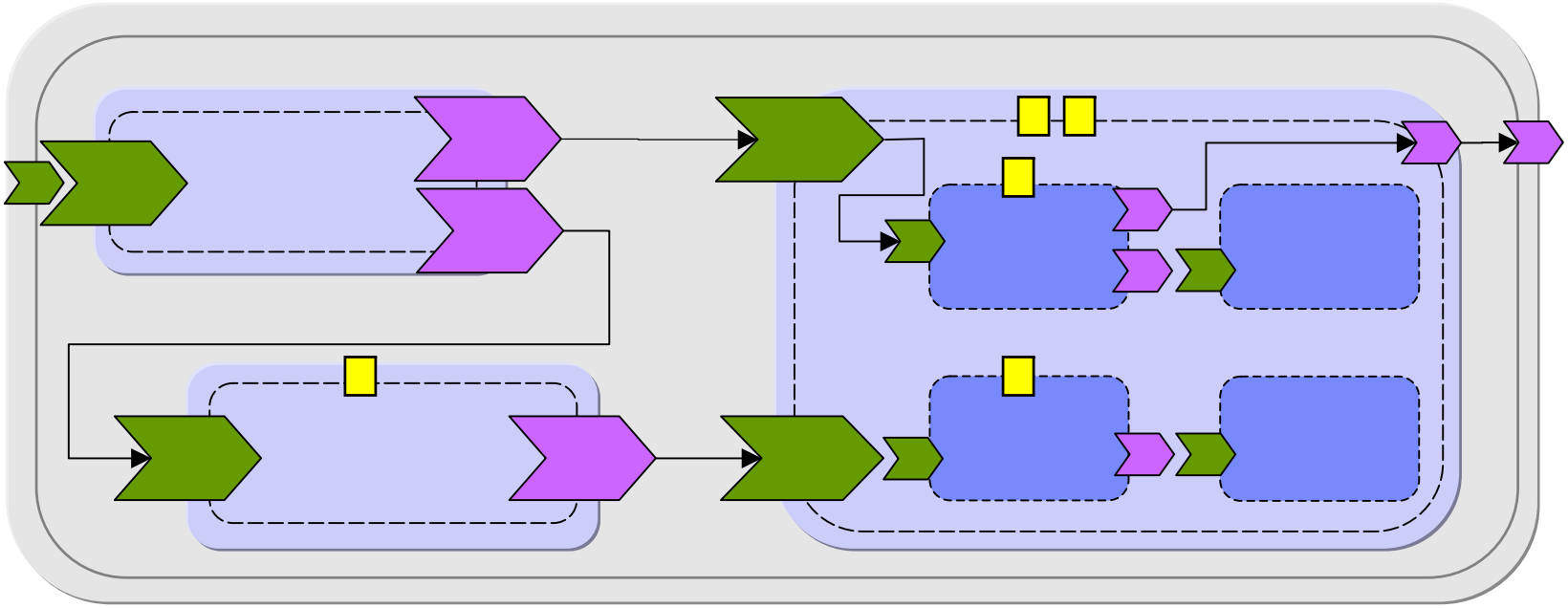
Übersicht

- **Einleitung: Von SOA zu SCA**
- **SCA Spezifikationen (Teil 1)**
  - Überblick
  - Assembly Model
  - Client and Implementation Model for Java
- **Tools und Live Demo**
- **SCA Spezifikationen (Teil 2)**
  - Binding & Policy Framework
- **Fazit**

SCA Spezifikationen

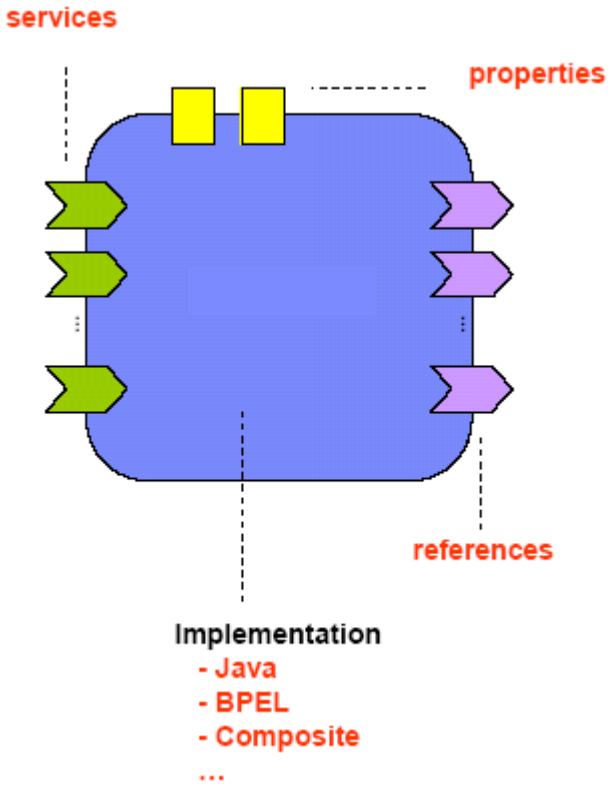
- **Assembly Model**
  - Vereinheitlichtes Modell für den Zusammenbau von Services (loose / stark gekoppelt)
    - Auflösen und Konfiguration von Abhängigkeiten
    - Deklarativer Ansatz (→ nicht im Implementierungs-Code)
- **Client and Implementation Models**
  - Spezifizieren Umsetzung von SCA Komponenten für eine konkrete Implementierungs-Sprache (Java, C++, BPEL, ...)
- **Binding and Policy Framework**
  - Anwenden von Infrastruktur-Funktionalitäten auf Services und Service-Interaktionen (Security, Transaktionen, ...)

Assembly Model: Bausteine



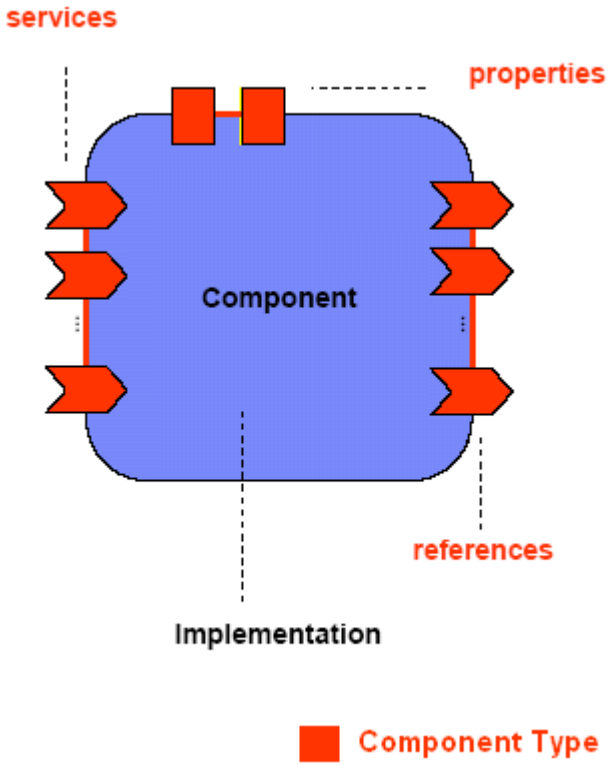
Assembly Model: Bausteine

- **Implementation**
  - Programmcode, der Business Logik konkret umsetzt
- **Service**
  - Angebot einer implementierten Business Funktion nach außen hin
- **Reference**
  - Abhängigkeit der Implementierung von einem externen Service
- **Property**
  - Konfigurierbarer Wert, der die Ausführung der Business Logik beeinflusst

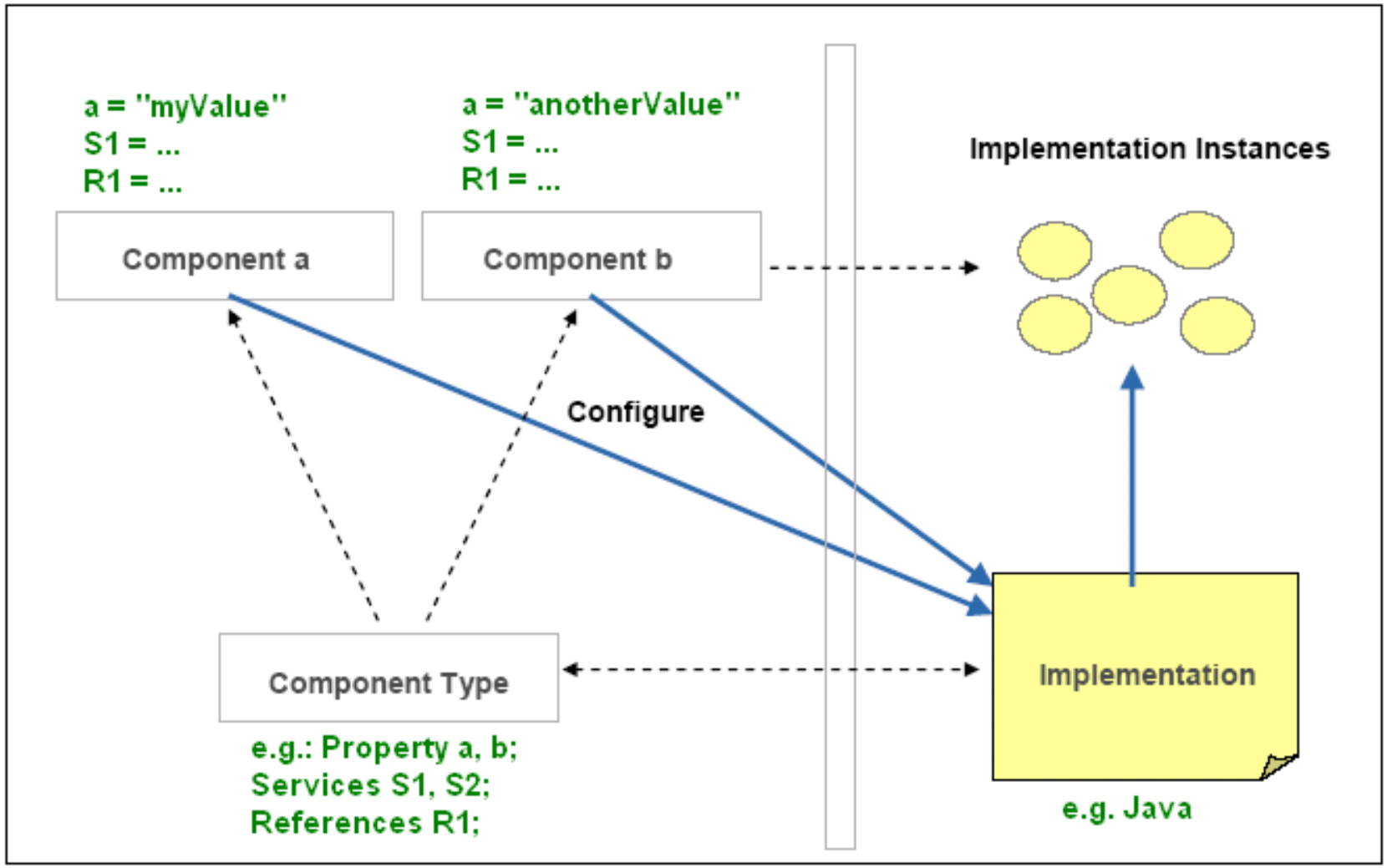


Assembly Model: Bausteine

- **Component Type**
  - Konfigurierbare Aspekte einer Implementierung
- **Component**
  - Konfigurierte Instanz einer Implementierung
    - Unterschiedlich konfigurierte Components mit jeweils derselben Implementierung möglich
- **Implementation Instance**
  - Laufzeit-Instanz einer konfigurierten Implementierung

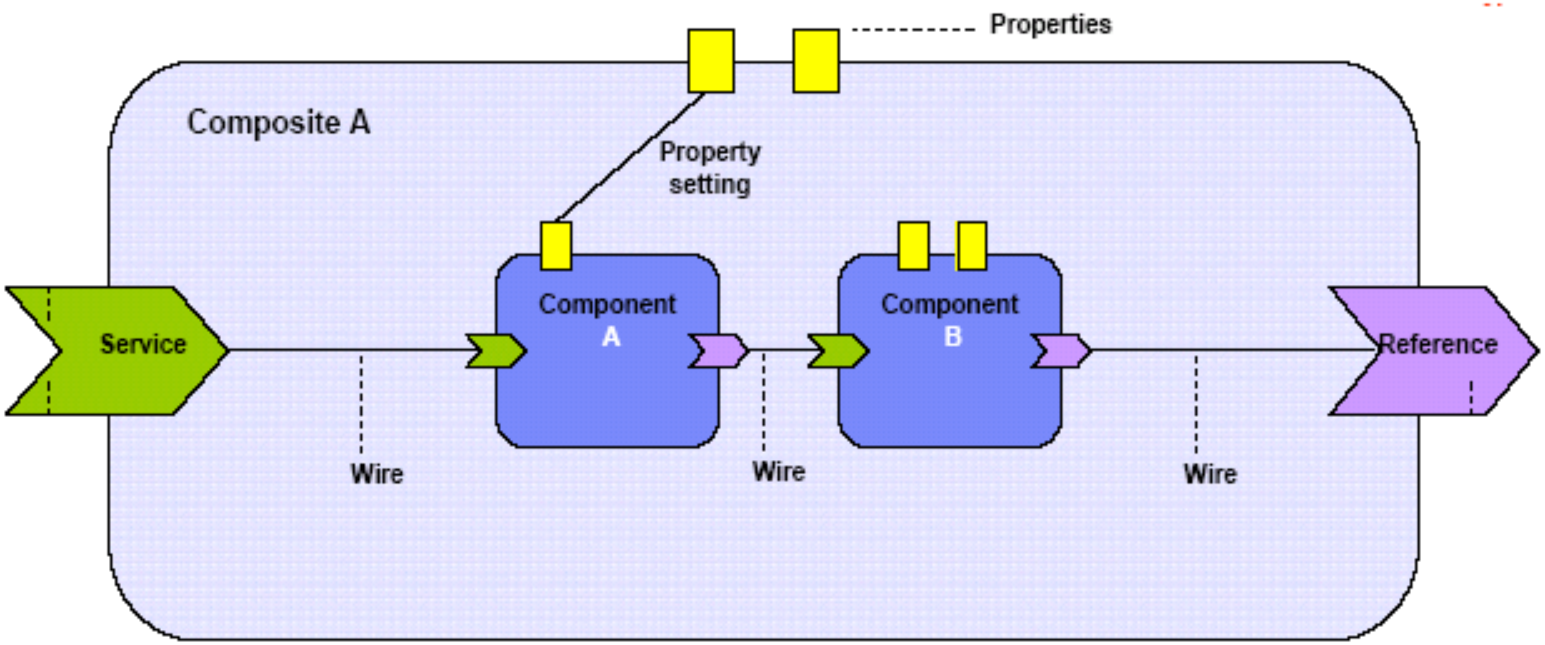


Mehrere Components pro Implementation



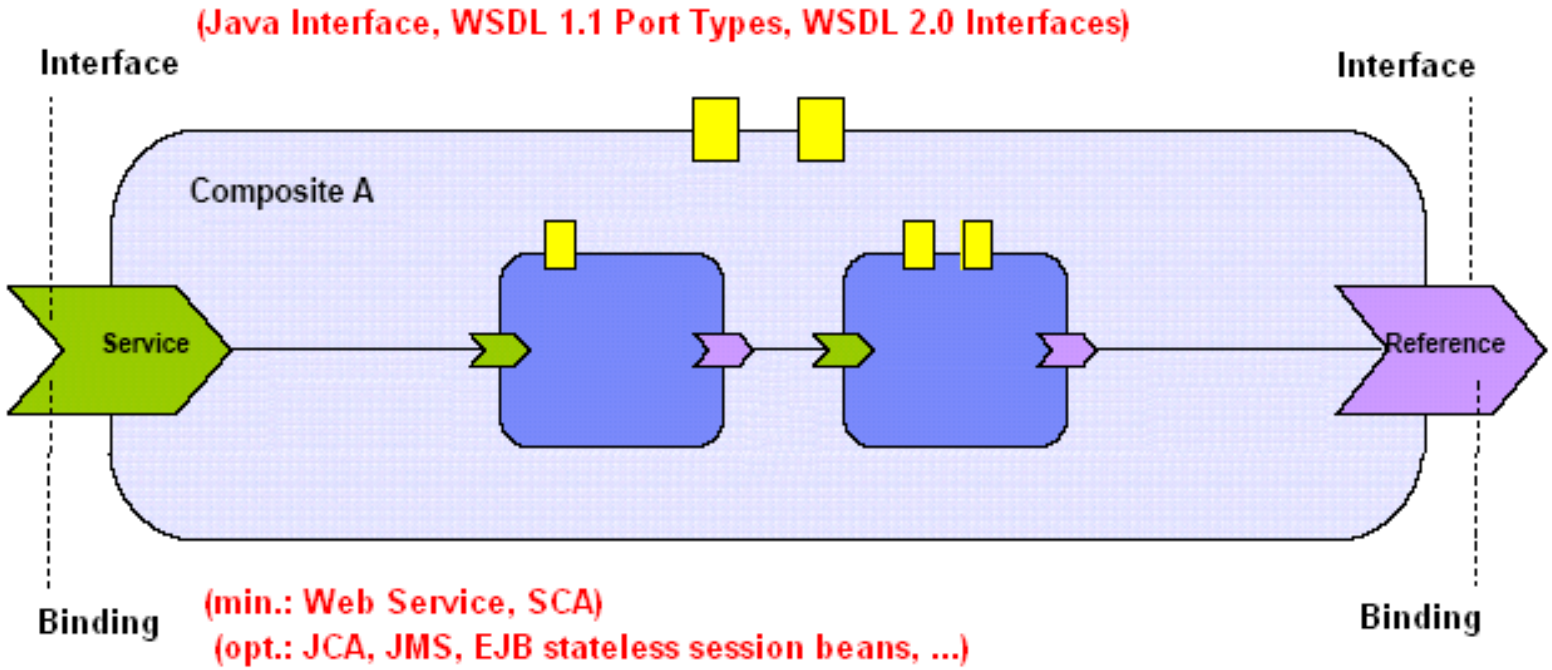
Assembly Model: Bausteine

- **Composite**
  - Grundbaustein für die Komposition von Components
- **Wire**
  - Verbindung (konfiguriert Services und References)

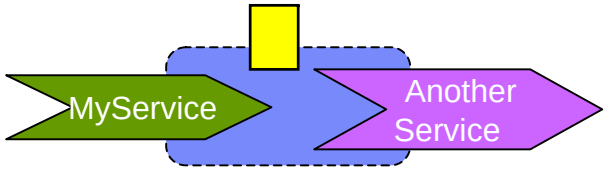


Services und References

- **Interface**
  - Beschreibung der Schnittstellen zu Business Funktionen
- **Binding**
  - Zugriffsart



Umsetzung des Assembly Models



- **Deklarative Beschreibung über XML-Deskriptoren**
  - Beispiel: Component Type

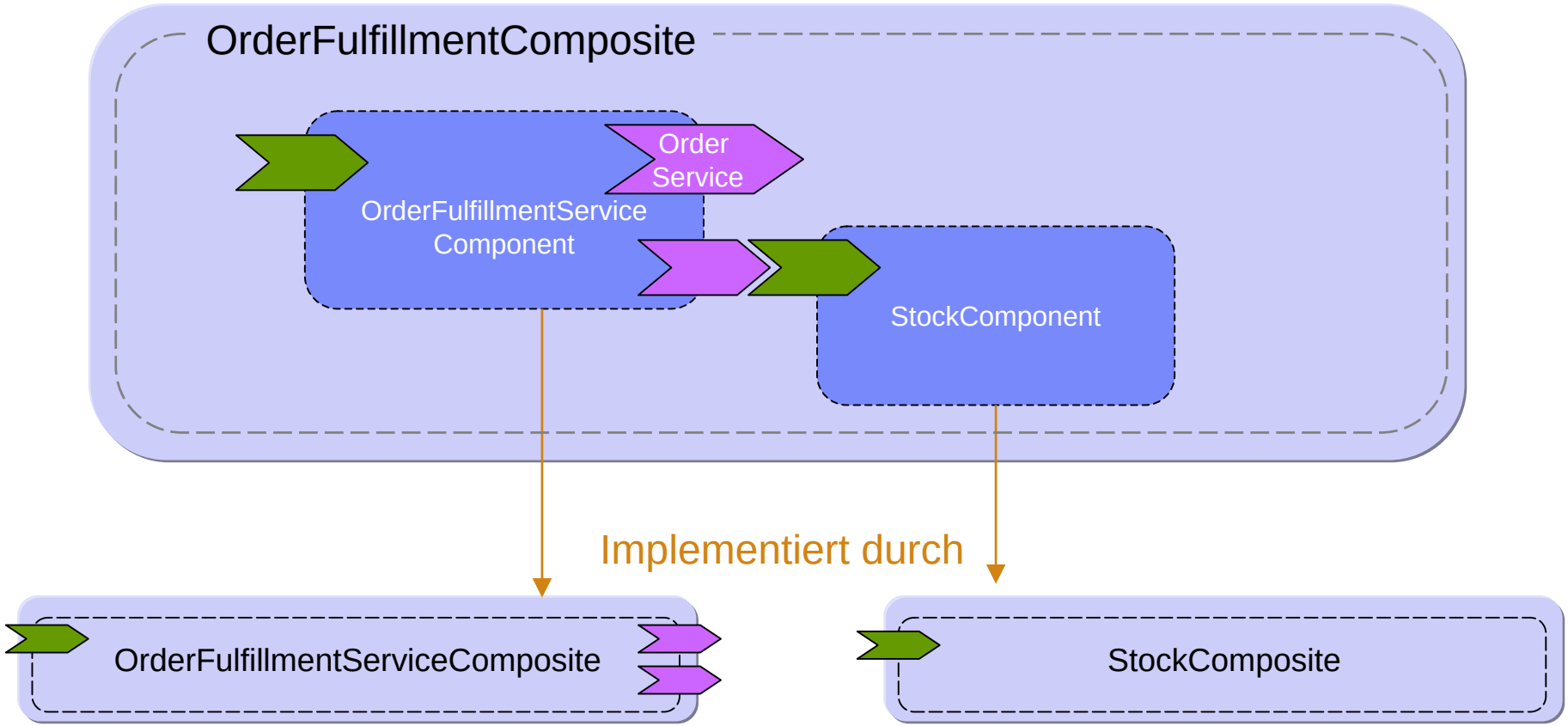
```
<?xml version="1.0" encoding="ASCII"?>
<componentType xmlns="http://www.osoa.org/xmlns/sca/1.0">
    <service name=„MyService">
        <interface.java interface="services.me.MyService"/>
    </service>
    <reference name=„AnotherService">
        <interface.java interface="services.dependencies.AnotherService"/>
    </reference>
    <property name="currency" type="xsd:string">EUR</property>
</componentType>
```

- **Dateinamen und -Struktur festgelegt von den SCA Spezifikationen**

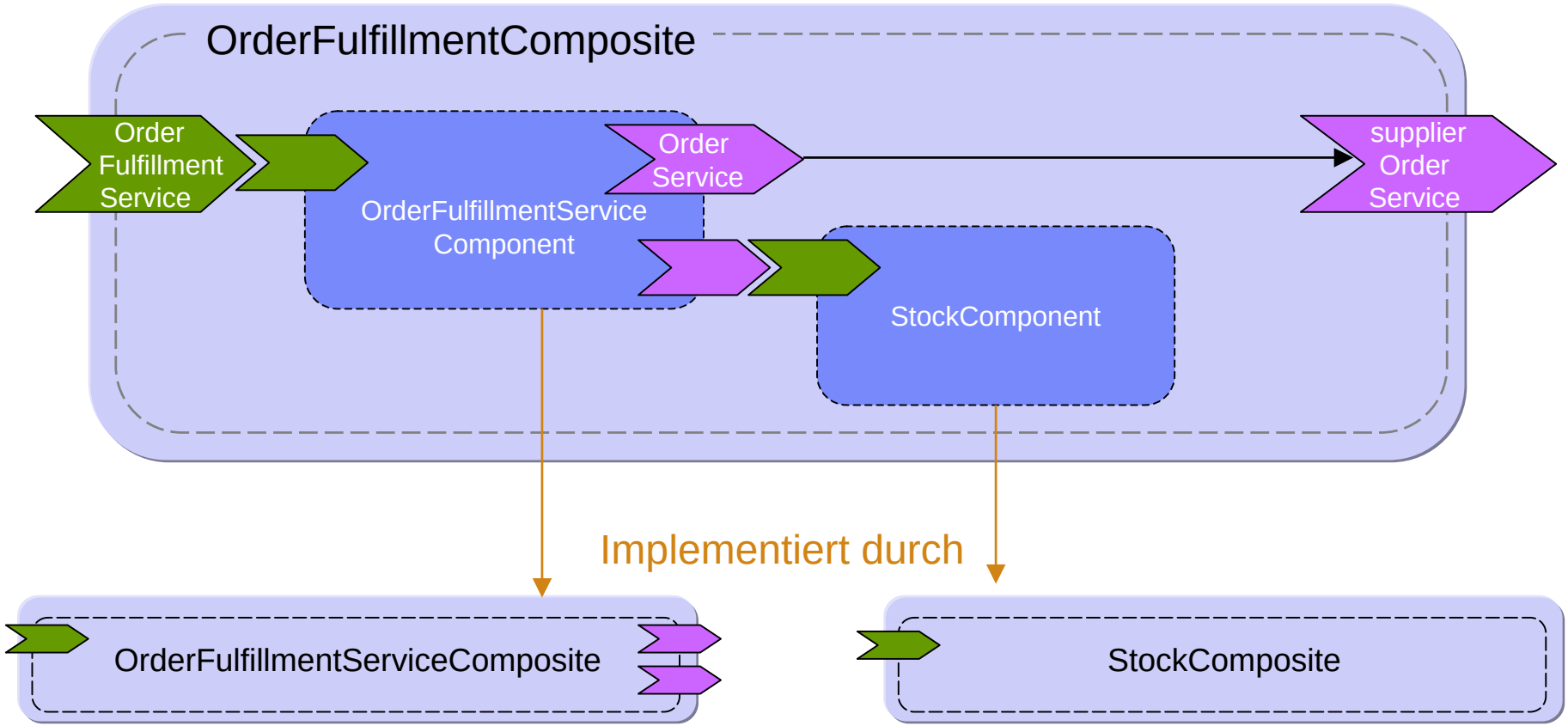
Umsetzung des Assembly Models, Beispiel

- **Situation:**
  - Shop
  - Bestellvorgang abschließen: Produkte besorgen
  - Zwei verschiedene Arten von Produkten
    - Eine davon immer intern auf Lager
    - Die andere müssen wir extern bestellen

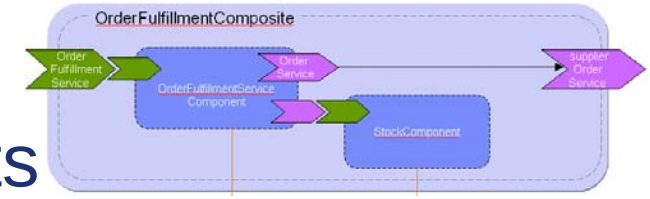
Umsetzung des Assembly Models, Beispiel: Composites als Components



Umsetzung des Assembly Models, Beispiel: Composites als Components

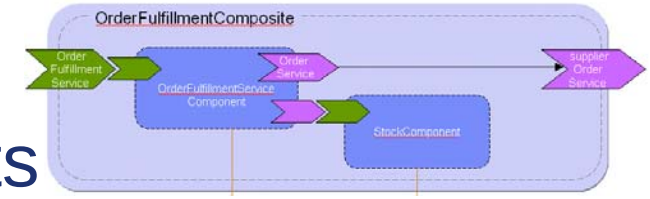


Umsetzung des Assembly Models, Beispiel: Composites als Components




```
<?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>
...
```

Umsetzung des Assembly Models, Beispiel: Composites als Components




```
...
  <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>
```

Umsetzung des Assembly Models, Composites Inclusion

- **Ermöglicht modulare Entwicklung von Composites**
- **Aufteilung eines Composites auf mehrere Dateien**
  - Einzelne Teile können in sich unvollständig sein
- **Textuelle Einbindung der Teile über <include/> Element:**

```
<?xml version="1.0" encoding="ASCII"?>
<composite xmlns=„http://www.osoa.org/xmlns/sca/1.0“
           name=„OrderFulfillmentComposite" >
   <include name=„OrderFulfillment_Services"/>
   <include name=„OrderFulfillment_Components"/>
   <include name=„OrderFulfillment_References"/>
   <include name=„OrderFulfillment_Wires"/>
</composite>
```

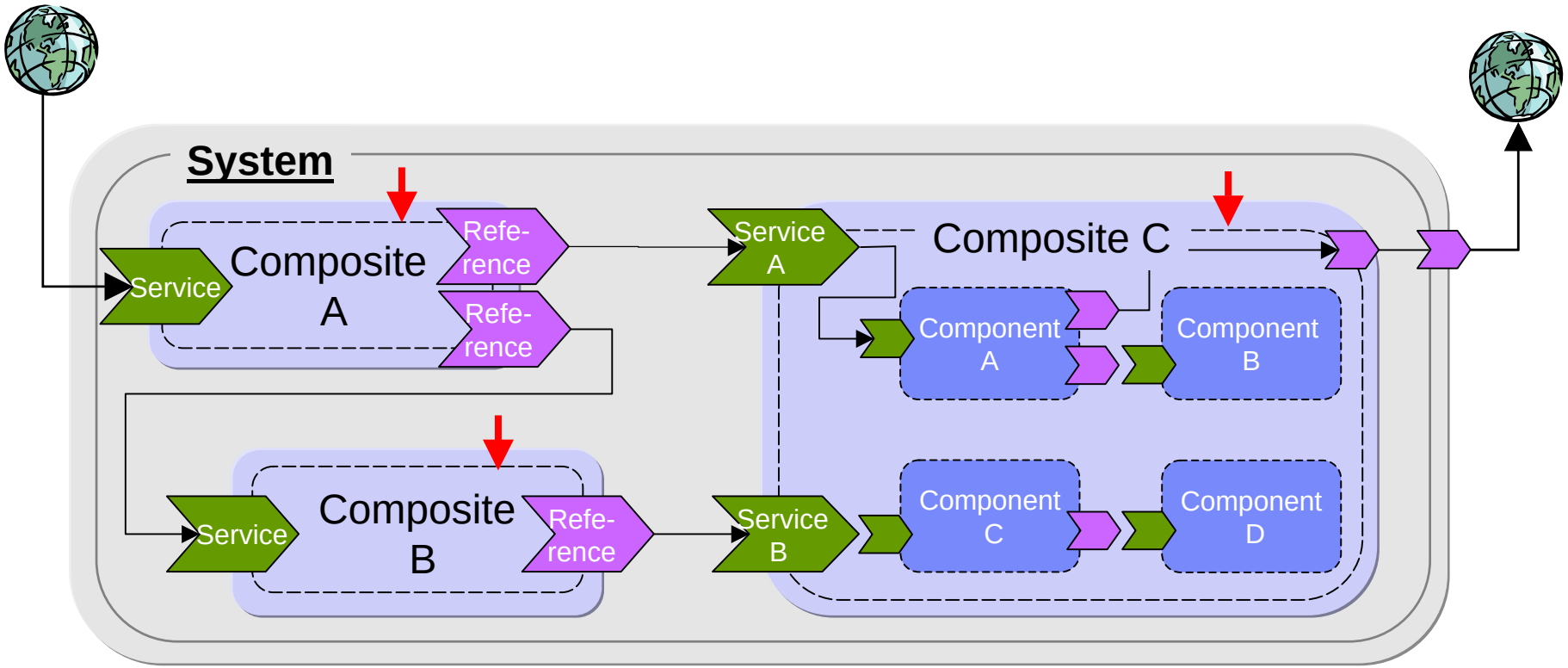
- **Resultierendes Composite muss vollständig und fehlerfrei sein (keine doppelten Definitionen)**

Deployment in einer SCA-Laufzeitumgebung

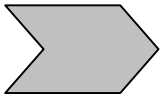
- **SCA System selbst verhält sich ähnlich wie ein Composite**
  - Top-Level Elemente für das Deployment sind Composites
  - Integration ins System über Includes
  - SCA System besteht also aus einer Menge von eingebundenen Composites
- **Konfigurations-Möglichkeiten beim Deployment**
  - Composites (Top-Level) dürfen teilweise unvollständig sein; ergänzende Konfiguration dann beim Deployment
  - Binding Adressen, Wires

Assembly Model: Bausteine

- **SCA System**



Interfaces: Remotable vs. Local



- **Remotable**
  - Gedacht für: *lose gekoppelte Interaktionen*
  - Semantik des Datenaustauschs: *immer by-value*
  - können von Clients aufgerufen werden, die auf einem anderen OS laufen
  - Method Overloading: *verboten*
- **Local**
  - Gedacht für: *starke gekoppelte Interaktionen*
  - Semantik des Datenaustauschs: *by-reference*
  - können NICHT von Clients aufgerufen werden, die auf einem anderen OS laufen
  - Method Overloading: *erlaubt*
- **Ziel der Unterscheidung: Performanz-Steigerung**

Interfaces: Bidirectional und Conversational

- **Relevant für asynchrone Aufrufe**
- **Bidirectional Interfaces**
  - Peer-to-Peer bidirektionale Verbindungen
  - Umsetzung: zusätzliche Definition eines Callback-Interfaces
- **Conversational Interfaces**
  - Sequenz von zusammengehörigen Operationen eines Services
  - Ein Zustand muss über mehrere Operationen gehalten werden
    - Typischerweise via Konversations-ID
    - Hier: Konversations-Management auf Applikations-Ebene
  - Unterstützung durch konkretes Binding (/Policy) nötig
    - Z.B. Web Service Binding mit WS-RM oder WS-Addressing, ...

Assembly Model: Erweiterbarkeit

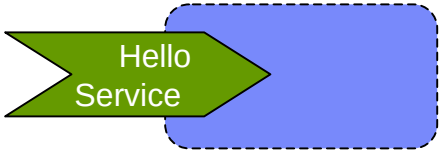
- **Interface Type**
  - Einschränkung: muss sich in WSDL umwandeln lassen
- **Implementation Type**
  - z.B. Ruby, JavaScript, XSLT, ...
- **Binding Type**

Client and Implementation Model for Java

- **Spezifiziert Abbildung auf Java von SCA Konzepten , u.a.**
  - Component, Service, Reference, Property, ...
  - Umsetzung von bidirectional und conversational Interfaces
  - JAVA API für SCA (für direkten Zugriff auf SCA Objekte)
- **Verdeutlicht das von SCA genutzte Konzept:**
  - Konfiguration durch Dependency Injection

Client and Implementation Model for Java

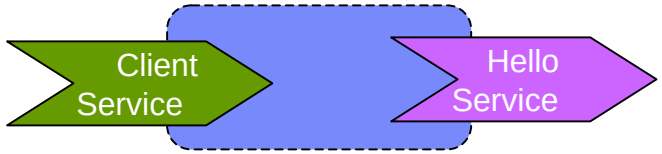
```
package services.hello;
import org.osoa.sca.annotations.*;

@Remotable
public interface HelloService {
    String hello(String message);
}
```



```
package services.hello;
import org.osoa.sca.annotations.*;

@Service(HelloService.class)
public class HelloServiceImpl implements HelloService {
    public String hello(String message) {
            ...
    }
}
```

Client and Implementation Model for Java

```
package services.client;
import services.hello.HelloService;
import org.osoa.sca.annotations.*;

@Service(ClientService.class)
public class ClientServiceImpl implements ClientService {

    private HelloService helloService;

    @Reference(name="helloService", required=true)
    public void setHelloService(HelloService service){
            helloService = service;
    }
    public String clientMethod() {
            String result = helloService.hello("Hello World!");
            ...
    }
}
```



Übersicht

- **Einleitung: Von SOA zu SCA**
- **SCA Spezifikationen (Teil 1)**
  - Überblick
  - Assembly Model
  - Client and Implementation Model for Java
- **Tools und Live Demo**
- **SCA Spezifikationen (Teil 2)**
  - Binding & Policy Framework
- **Fazit**

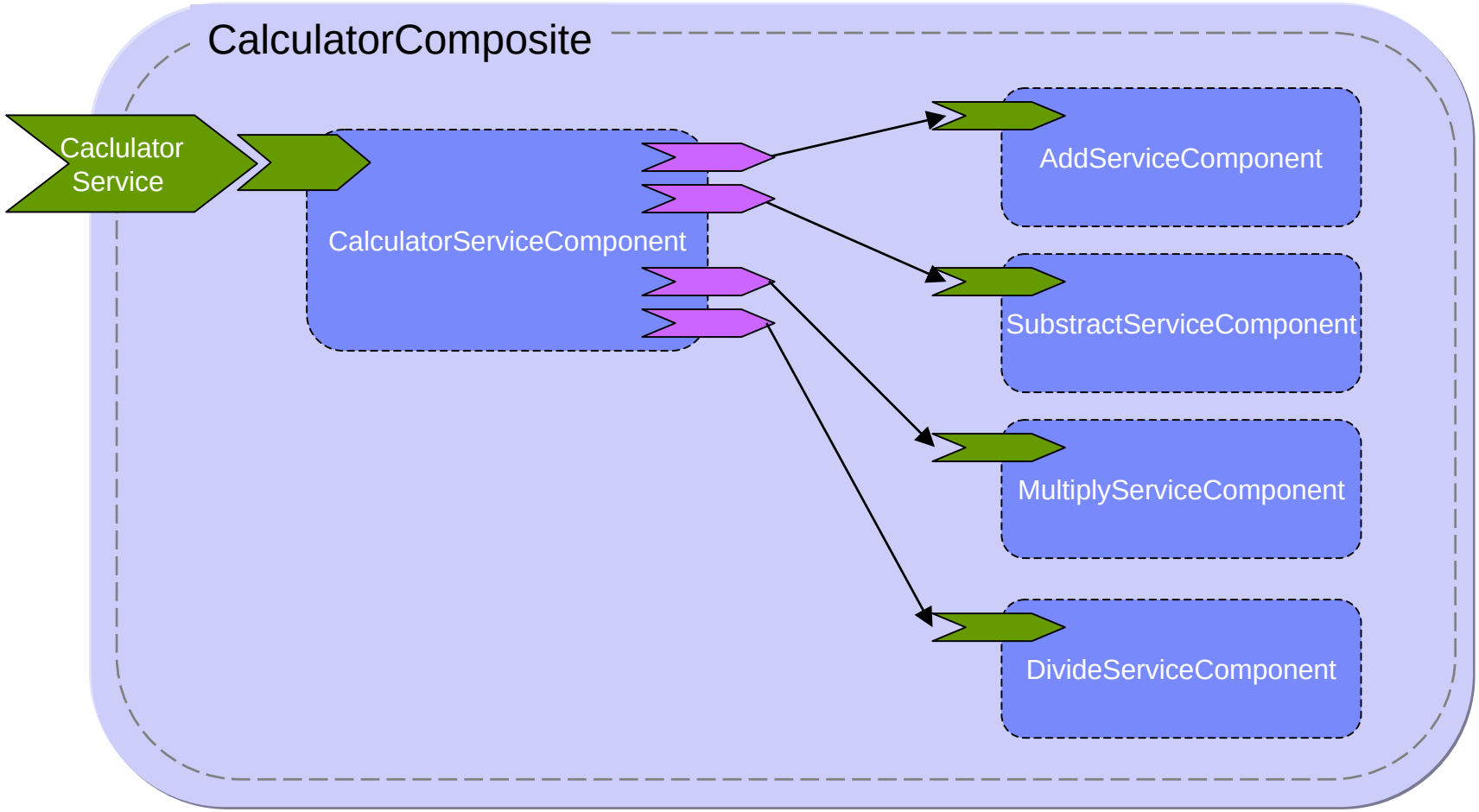
SCA Implementierungen & Tools

- **Kommerziell**
  - IBM (WebSphere App.Server), Oracle (EDA Suite), ...
- **Open Source**
  - Apache Tuscany
    - Implementierung (Laufzeitumgebung) für Java und C++
  - SOA PHP Project (PHP PECL SCA/SDO)
    - Implementierung (Laufzeitumgebung) für PHP
  - Eclipse SOA Tools Platform Project (STP)
    - Ziel: Entwicklung von diversen Tools im SOA Umfeld
    - „Core“ Subprojekt mit Assembly Model

Apache Tuscany

- **Apache Projekt**
  - „Incubator“ Status (Release M2)
- **3 Teile**
  - Service Component Architecture (SCA)
    - Einheitlicher Zugriff auf Services
  - Service Data Objects (SDO)
    - Einheitliches Verarbeitungmodell für ausgetauschte Daten
  - Data Access Services (DAS)
    - Einheitlicher Zugriff auf Daten

Apache Tuscany: Live Demo



Übersicht

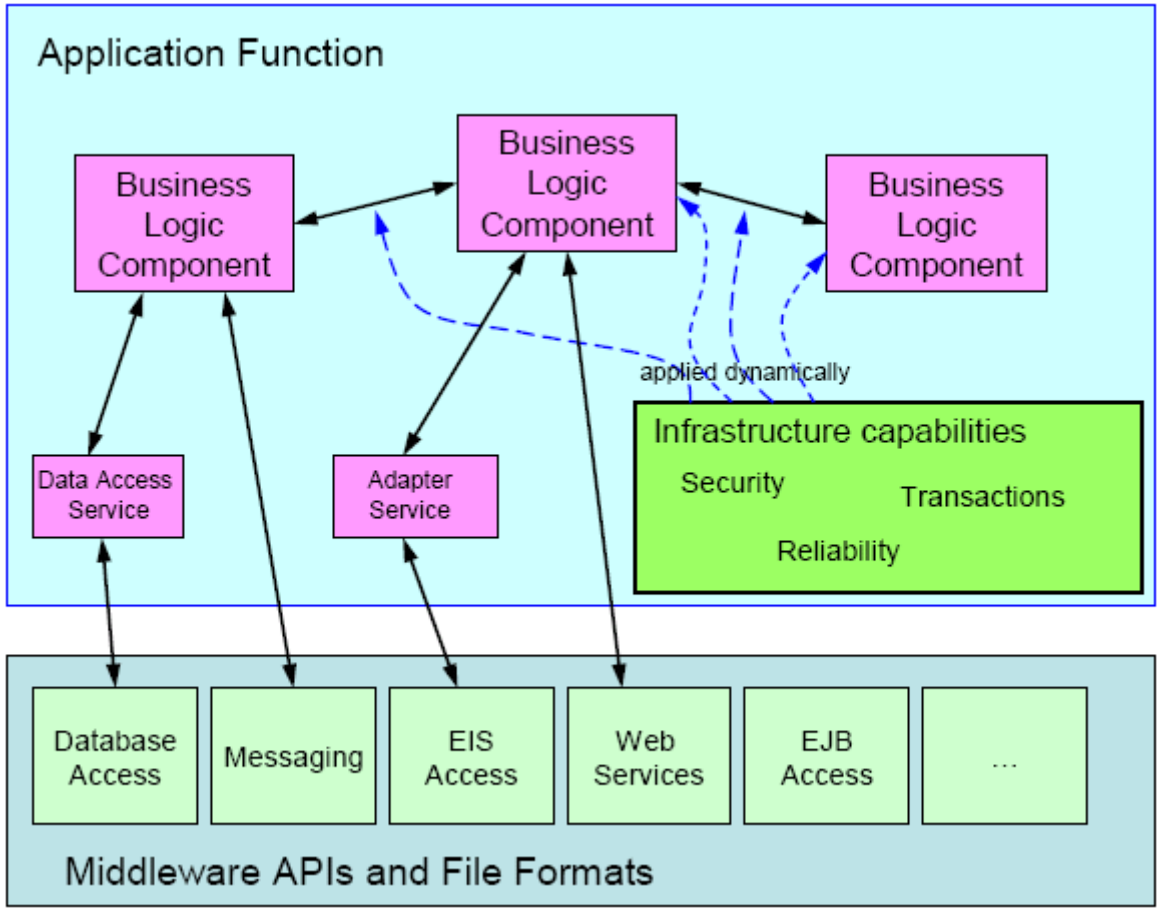
- **Einleitung: Von SOA zu SCA**
- **SCA Spezifikationen (Teil 1)**
  - Überblick
  - Assembly Model
  - Client and Implementation Model for Java
- **Tools und Live Demo**
- **SCA Spezifikationen (Teil 2)**
  - Binding & Policy Framework
- **Fazit**

Binding & Policy Framework

- **Spezifizieren von nichtfunktionalen Aspekten**
  - Anforderungen
  - Fähigkeiten
  - Quality-of-Service
- **OSOA: Schwerpunkt zunächst auf**
  - Security
  - Reliability
  - Transactions
- **Warum ist das im SCA Modell berücksichtigt?**
  - Flexibilität: Konfigurationen für verschiedene Umgebungen

Integration von Policies in SCA

- **Prinzip: anhängen von Policies an SCA Elemente**



Integration von Policies in SCA

- **Interaction Policies**
  - Anwendung auf Services / References
  - Bezug: Kommunikation zwischen Service Anbieter und Client
  - Beispiel: Vertraulichkeit
- **Implementation Policies**
  - Anwendung auf Components
  - Bezug: Fähigkeiten, die der Container bereitstellen soll, in dem ein Component läuft
  - Beispiel: Component muss in einer Transaktion laufen

Binding & Policy Framework: Bausteine

- **Intents**
  - Abstrakte, high-level Anforderungen
  - Unabhängig von Implementierungs-Technologie / Binding
  - Beispiel: „confidentiality“ oder „confidentiality/high“
- **Policy Sets**
  - Mapping von konkreten Policies auf Intents
  - Angabe beim Deployment oder direkte Definition in einem Binding- bzw. Component-Element
- **Konkrete Policies**
  - via WS-Policy, WS-PolicyAttachment
  - theoretisch auch andere Möglichkeiten vorgesehen
- **...**

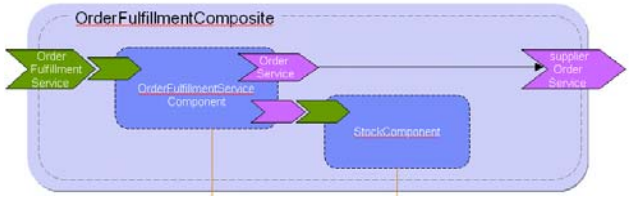
Beispiele

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

-----------------------------------------------------------------------

<sca:reference name="FlightService">
        <sca:interface … />
        <sca:binding.WS policySet =”BasicSecurity”/>
</sca:reference>
```

Übersicht

- **Einleitung: Von SOA zu SCA**
- **SCA Spezifikationen (Teil 1)**
  - Überblick
  - Assembly Model
  - Client and Implementation Model for Java
- **Tools und Live Demo**
- **SCA Spezifikationen (Teil 2)**
  - Binding & Policy Framework
- **Fazit**

Zusammenfassung



- **SCA Modell soll Flexibilität, Wiederverwendbarkeit und Integrationsfähigkeit erhöhen**
  - Modulare Komposition von Komponenten, verschiedene Implementierungs-Technologien
  - Wiederverwendung durch unterschiedliche Konfigurationen einer Implementierung
  - Trennung von Business Logik, Middleware und nichtfunktionalen Aspekten (Separation of Concerns); konkrete Technologie-Entscheidungen durch Binding
- **SCA Open Source Software?**

Technologien im Umfeld von SCA

- **SCA – verbundene Technolgien**
  - Service Data Objects (SDO), Data Access Objects (DAS)
- **Wo war Microsoft bei Open SOA und SCA?**
  - Windows Communication Foundation (WCF / „Indigo“)
    - seit 2003
    - Gemeinsamkeiten mit SCA in vielen Punkten, aber:
      - rein .NET basiert
      - keine Assembly von Komponenten, kein Wiring
      - kein Focus auf speziellen Technologien für Datenaustausch zwischen Komponenten (bei SCA: SDO)

Stimmen

“SCA has the potential to provide significant value in the Java world and beyond. If the vendors behind this new technology can complete the tasks they’ve set for themselves, we can look forward to a day when the two major foundations for creating service-oriented applications are SCA and WCF.”

Foundations for Service-Oriented Applications: Comparing WCF and SCA
David Chappell, December 2005

“Service Component Architecture has the potential to significantly aid mainstream organizations in the development, deployment and management of services using a service-oriented architecture.”

SCA Is a Winner in the Quest to Establish a Common Notation for SOA
Jess Thompson, Gartner, March 2006

Vielen Dank für die Aufmerksamkeit...

Fragen?