



JavaOne

Spring and Service Component Architecture as the Basis for Distributed Services Applications

Adrian Colyer, CTO, Interface 21

Mike Edwards, Strategist, IBM

www.osoa.org

www.springframework.org

TS-8194

Aim of This Session

Learn how Spring and SCA
combine to create distributed
service-based applications

Agenda

SCA Overview

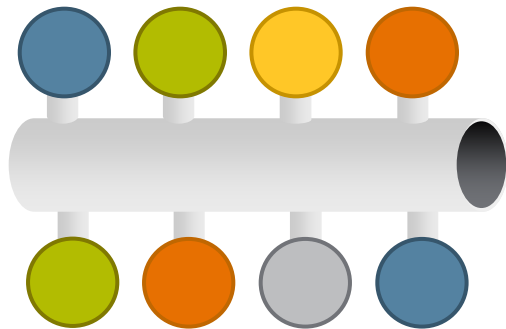
Spring Framework

Building Composite Services Applications

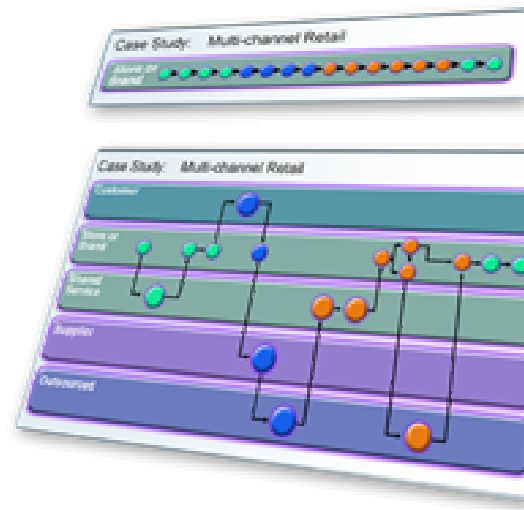
Demo

What We Want to Get To

- Well-defined interfaces with business-level semantics
- Standardized communication protocols
- Flexible recombination of services to enhance software flexibility



+



Service-Oriented Architecture is one of the key technologies to enable flexibility and reduce complexity

The SOA Programming Model

- SOA Programming Model derives from the basic concept of a **service**
 - A service is an abstraction that encapsulates a software function
- **Developers** build services, use services, and develop solutions that aggregate services
 - **Composition** of services into integrated **solutions** is a key activity

The SOA Programming Model

- Core elements
 - **Service Assembly**
 - Technology- and language-independent representation of composition of services
 - **Service Components**
 - Technology- and language-independent representation of composable service implementation
 - **Service Data Objects**
 - Technology- and language-independent representation of service data entity

What Is SCA?

- **Executable** model for assembly of service components into business solutions
- Simplified **component programming model** for implementation of services
 - Business services implemented in any of a variety of technologies
 - e.g., EJB™ beans, Java™ platform POJOs, BPEL process, COBOL, C++, PHP...

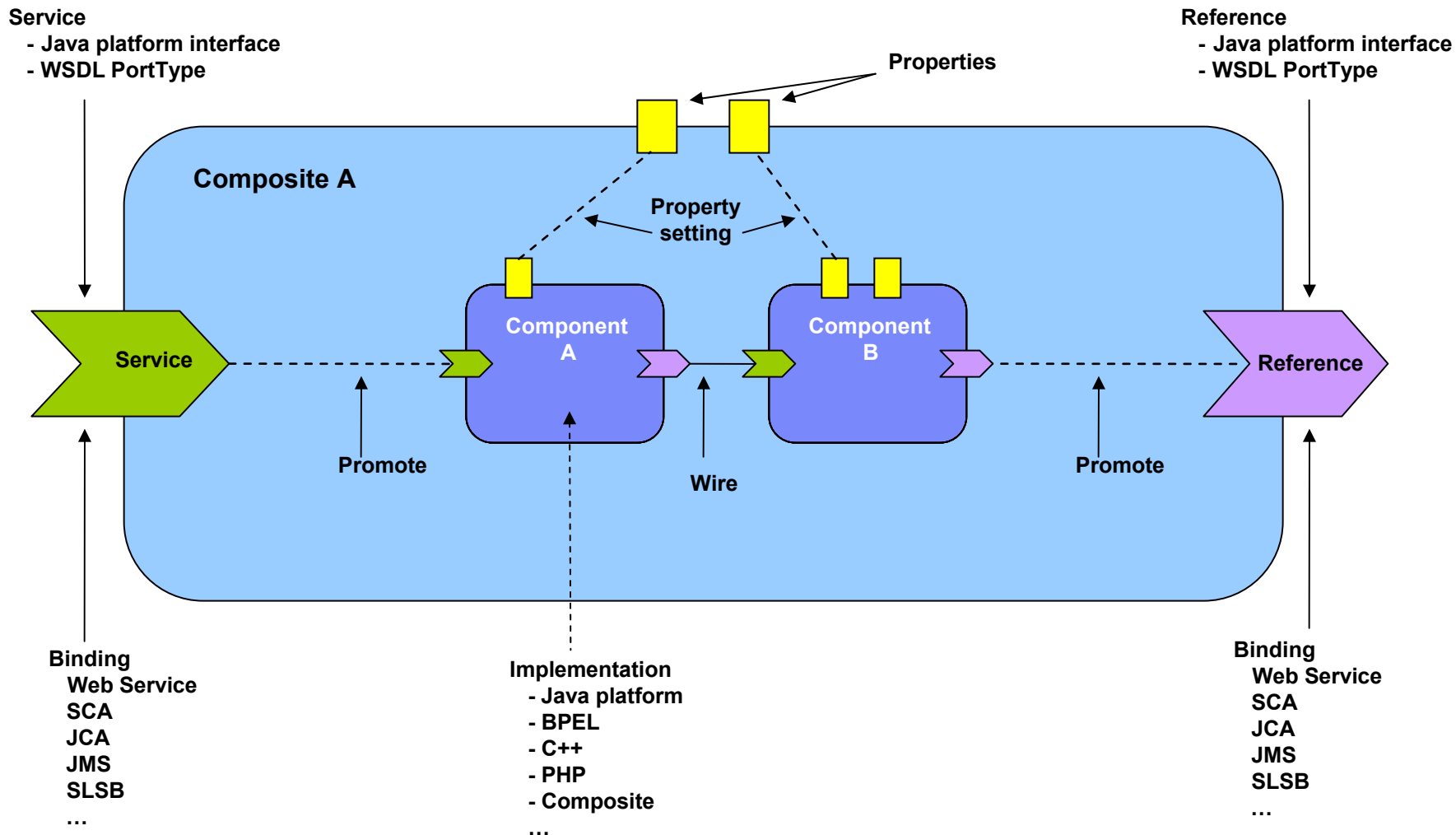
Key Benefits of SCA

- **Loose Coupling:** Components integrate without needing to know how other components are implemented
 - KEY requirement for SOA
- **Flexibility:** Components can easily be replaced by other components
 - KEY requirement for SOA
- **Services** can be easily invoked either synchronously or asynchronously
- **Composition** of solutions: clearly described
 - KEY requirement for SOA
- **Productivity:** Easier to integrate components to form composite application

Key Benefits of SCA

- SCA simplifies development experience for **all** developers, integrators, and application deployers

SCA Composite



JCA = "Java Connector Architecture" JMS = "Java Message Service"

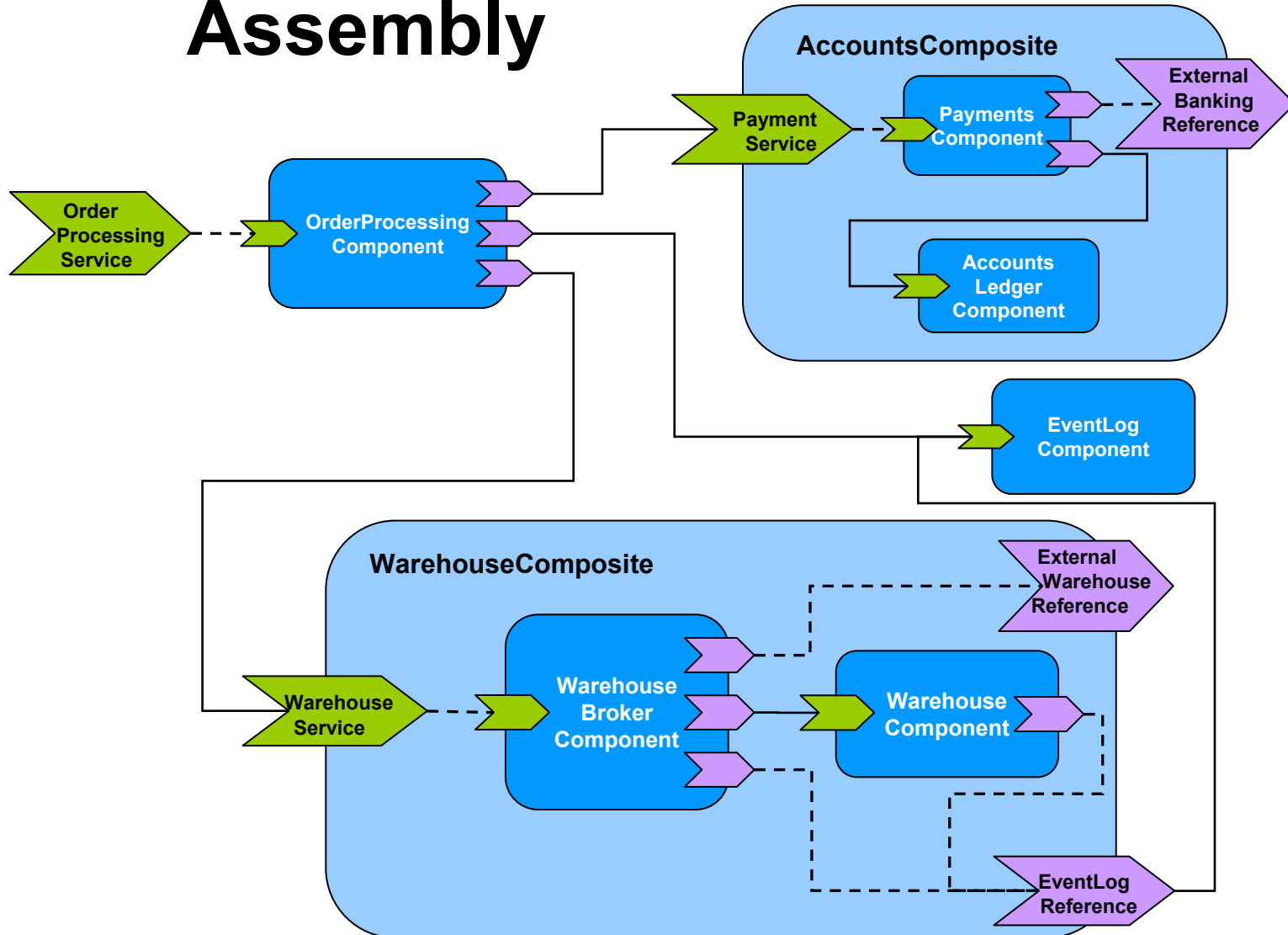
SCA Thumbnail Sketch

- **Component**
 - Configured instance of implementation
 - Provides and consumes services
 - Sets implementation properties
- **Composite**
 - Combines collections of components
 - Wires references to services
 - Selects bindings, endpoints
 - Applies policies

SCA Characteristics

- Distributed systems
- Heterogeneous systems
 - Multiple implementation languages
 - Different languages/different frameworks
 - Multiple communication mechanisms
- Declarative application of infrastructure service
- “Keep APIs out of the business logic”
 - Philosophy for component implementation

Example SCA Assembly



Agenda

SCA Overview

Spring Framework

Building Composite Services Applications
Demo

What Is the Spring Framework?

- A **lightweight container** for application objects
 - Lifecycle management
 - Dependency management and wiring
 - Non-invasive enterprise services
- A **set of modules** simplifying common enterprise application development tasks
 - Persistence
 - Messaging
 - ...
- An **approach** to building enterprise applications

The Spring Approach

- Program simply using objects
 - a.k.a. “POJOs”
 - Non-invasive
- Retain architectural choice
 - Keep environmental assumptions and dependencies out of application objects
- Facilitate test-driven development

Spring Is Responsible for...

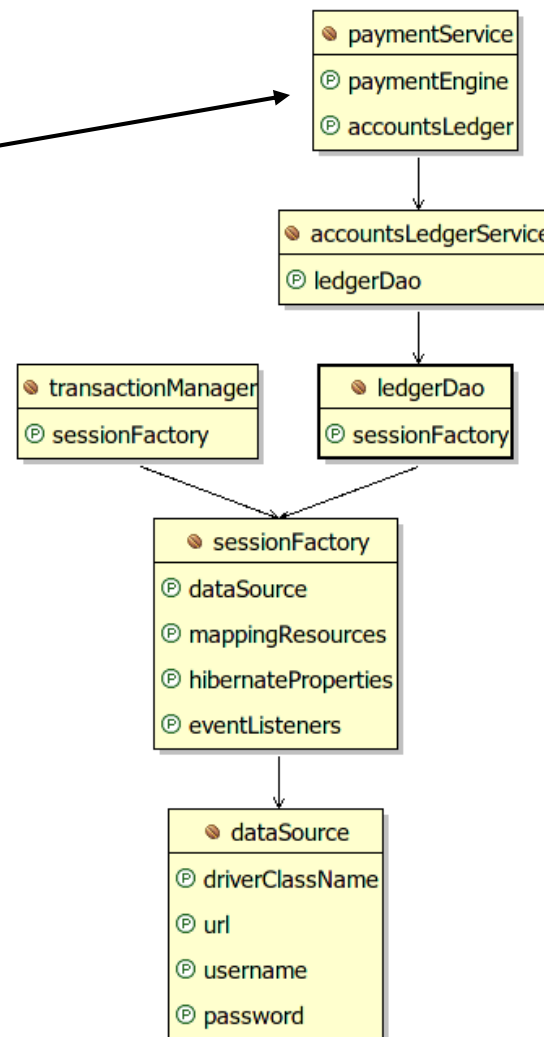
- **Instantiating** component instances
 - Account service, account dao, session factory, data source, etc.
- **Configuring** component instances
 - Setting simple properties
- **Decorating** components
 - Ensuring enterprise services such as transaction management are in place
- **Assembling** components into a fully functioning application
 - Wiring components together so that they can do their jobs

Sample Configurati

```

<bean id="paymentService"
  class="...PaymentServiceImpl">
  <property
    name="paymentEngine"
    ref="paymentEngine"/>
  <property
    name="accountsLedger"
    ref="accountsLedgerService"/>
</bean>

<bean ... />
  
```



Agenda

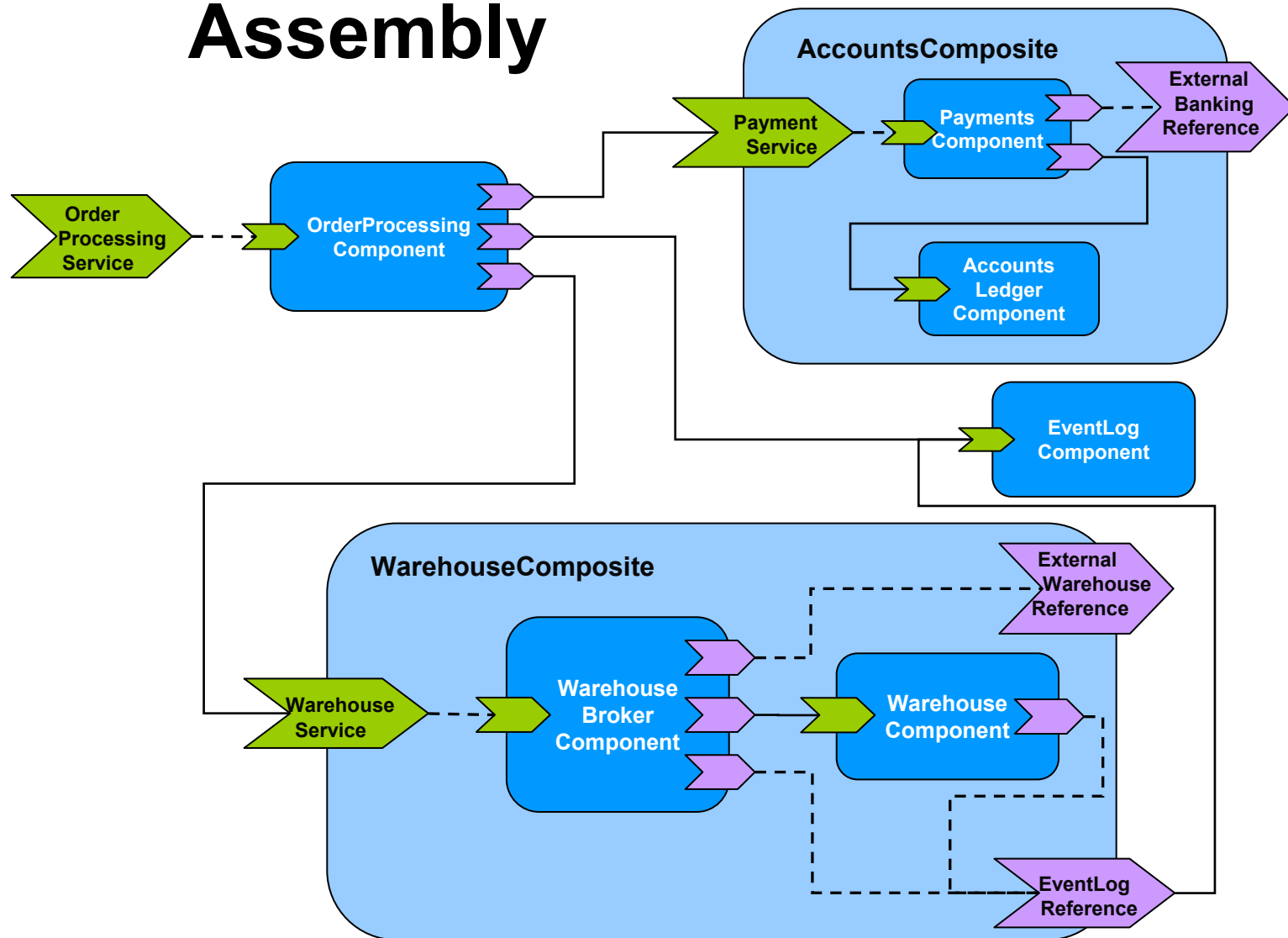
SCA Overview

Spring Framework

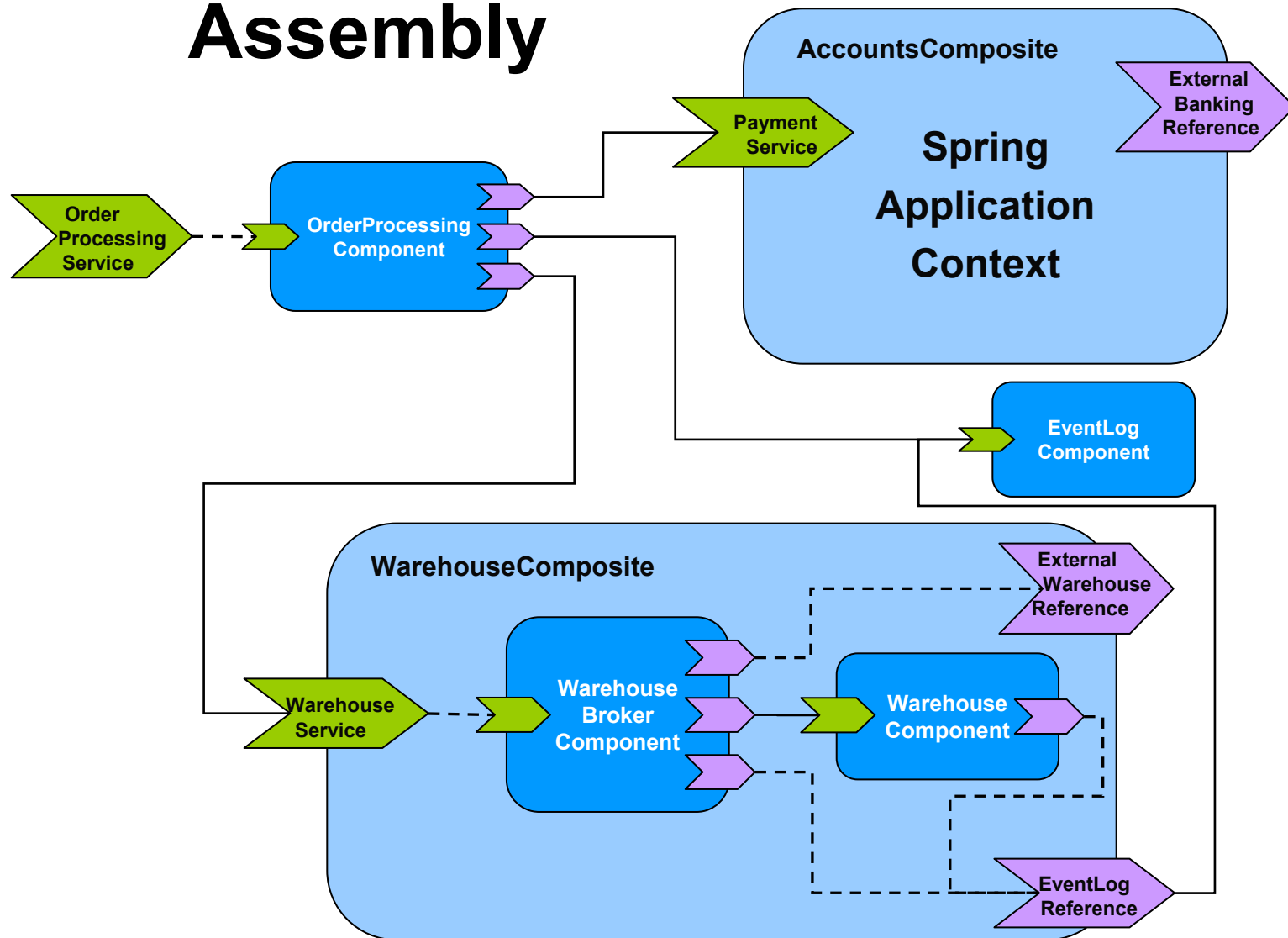
Building Composite Services Applications

Demo

Example SCA Assembly



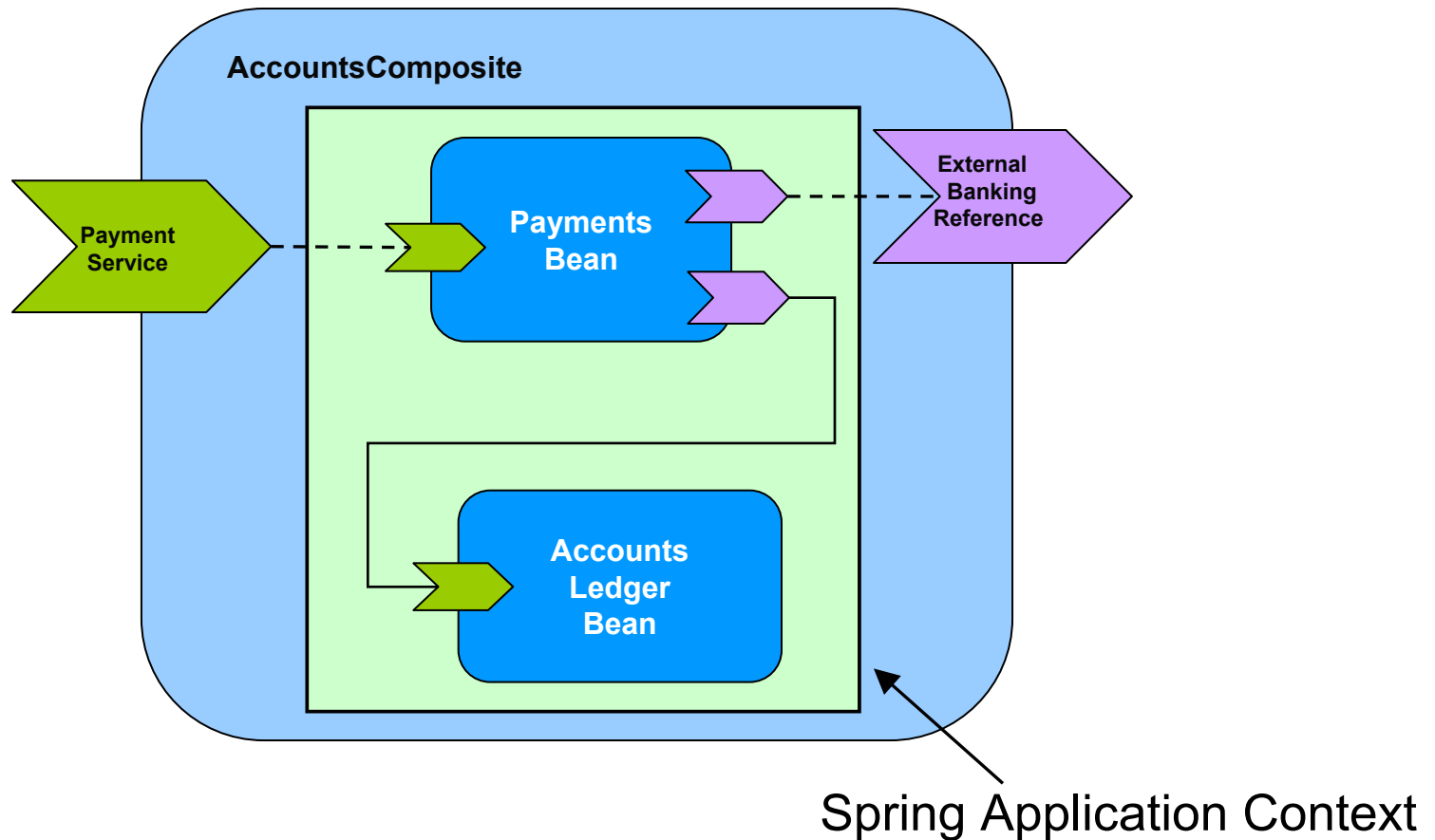
Example SCA Assembly



Spring-Powered Composites

- Spring can be used to implement SCA composites
 - Spring beans can be **exposed to** other SCA components
 - Spring beans can be **injected with** references to SCA components
 - Exposed services and supplied references can have infrastructure capabilities applied via SCA metadata
- Easily **integrate** existing spring application into an SOA
 - Spring is a proven model for building enterprise applications
 - SCA enables those applications to be integrated into a larger whole

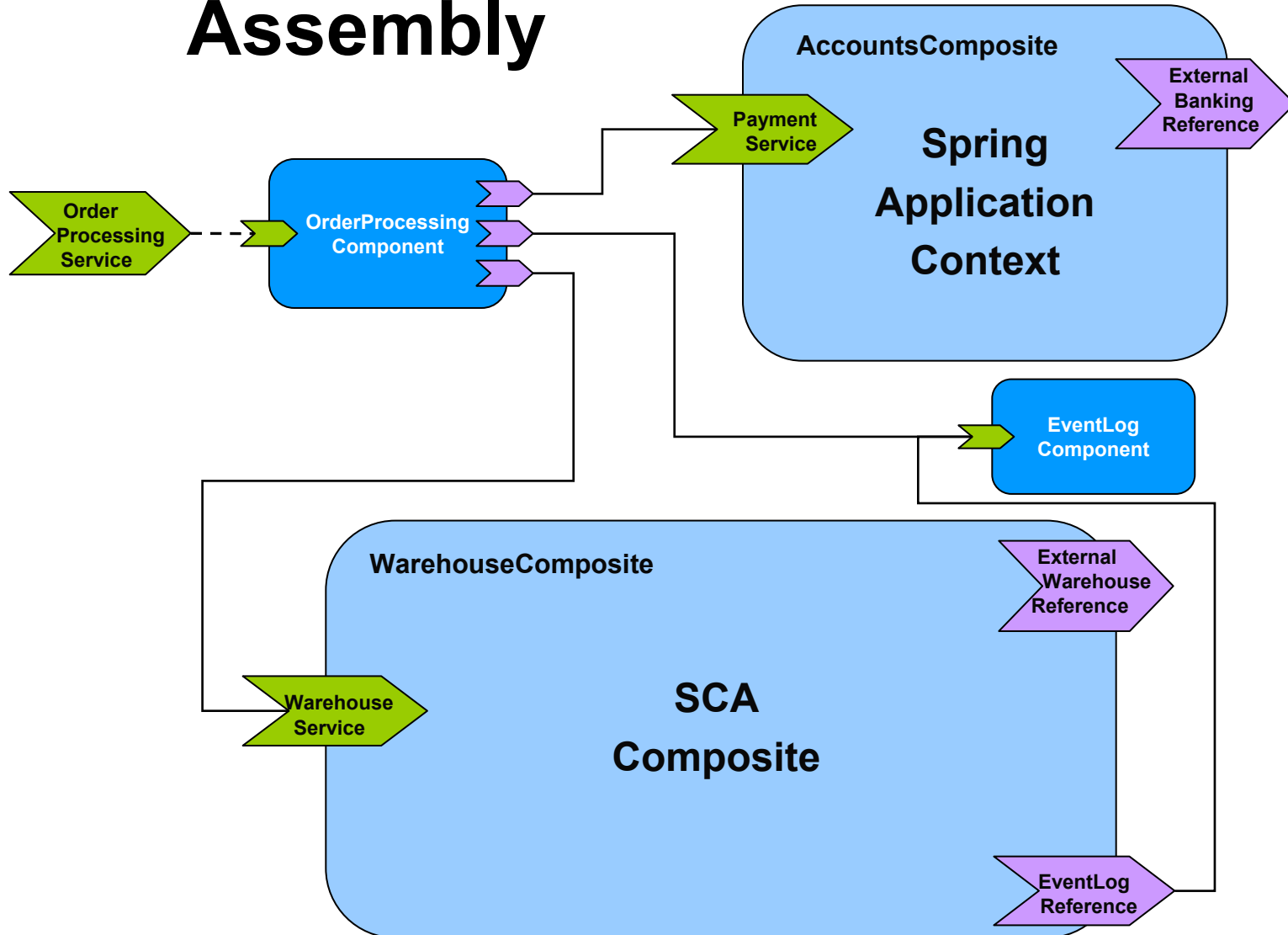
Spring-Powered Composites



Introducing SCA Composite Documents

- SCA Composite documents
 - XML representation of composites
- `<composite../>`
 - `<service.../>`
 - `<reference...>`
 - `<component.../>`
- Provide attachment points for bindings, endpoints, policies
- Specifies wires between references and services

Example SCA Assembly



Example Composite Document (1)

```
<?xml version="1.0" encoding="UTF-8"?>

<composite name="OrderProcessingComposite"

    xmlns:sca="http://www.osoa.org/xmlns/sca/1.0"

    xmlns:foo="http://www.foo.com/services"

    <!-- The Order Processing Service , offered as a Web service -->

    <service name="OrderProcessing" promote="OrderProcessing">

        <interface.wsd1 interface="http://www.foo.com/OrderProcessing#

            wsdl.interface(OrderProcessingExternal) "/>

        <binding.ws/>

    </service>

    <!-- The Order Processing Component -->

    <component name="OrderProcessing">

        <implementation.java class="com.foo.OrderProcess"/>

        <!-- Wires for the references of the component -->

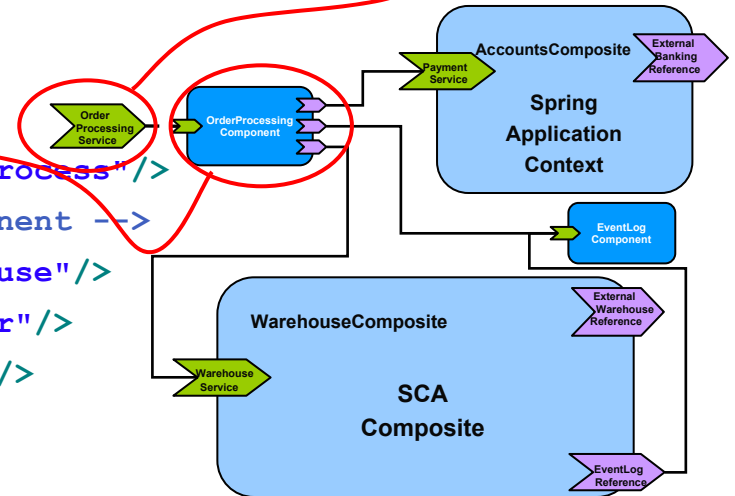
        <reference name="Warehouse" target="Warehouse"/>

        <reference name="Customer" target="Customer"/>

        <reference name="Events" target="EventLog"/>

    </component>


```



Example Composite Document (2)

```
<!-- The AccountsSystem application modeled as single Spring application context -->
```

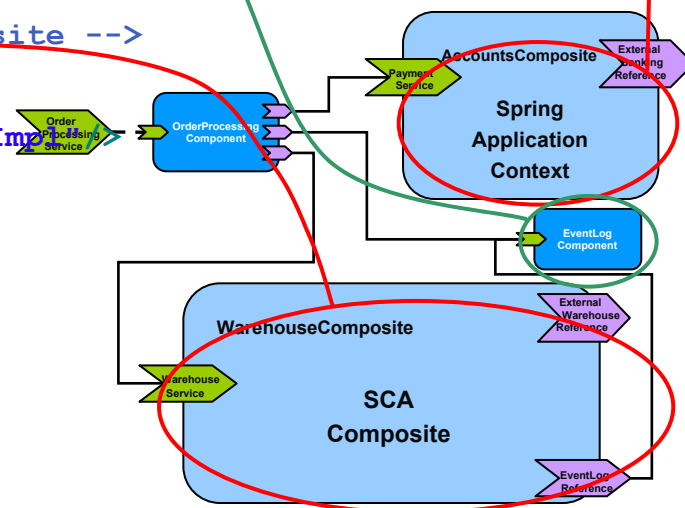
```
<component name="AccountsSystem">
    <implementation.spring location="AccountsSystemImpl.jar"/>
    <service name="PaymentService"/>
    <reference name="Banking"/>
</component>
```

```
<!-- The EventLog component is modeled as a single Java POJO -->
```

```
<component name="EventLog">
    <implementation.java class="com.foo.EventLoggerImpl"/>
</component>
```

```
<!-- The Warehouse component - implemented as a composite -->
```

```
<component name="Warehouse">
    <implementation.composite name="foo:WarehouseImpl">
        <service name="WarehouseService"/>
        <reference name="Events" target="EventLog"/>
    </implementation.composite>
</component>
```



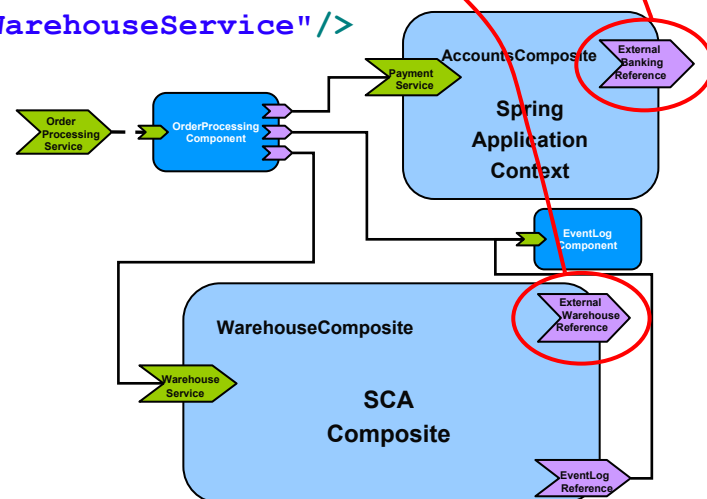
Example Composite Document (3)

```

<!-- Reference to external banking service using Web service binding -->
<reference name="ExternalBanking" promote="AccountsSystem/Banking">
  <interface.wSDL interface="http://www.foo.com/Accounts#
    wsdl.interface(ExternalBanking)"/>
  <binding.ws uri="http://www.hugebank.com/BankingService"/>
</reference>

<!-- Reference to external Warehouse service using Web service binding -->
<reference name="ExternalWarehouse" promote="Warehouse/Warehouse2">
  <interface.wSDL interface="http://www.foo.com/Warehouse#
    wsdl.interface(ExternalWarehouse)"/>
  <binding.ws uri="http://www.bigwarehouse.com/WarehouseService"/>
</reference>

</composite>
  
```



Things to Note

- No need to specify bindings for internal wires
 - Defaults to “binding.sca” which the runtime can provide
- No need to specify endpoints for internal services
 - Allocated by SCA runtime
- Identify wire endpoints through “component/service”
 - Can omit service name if only one service...

Spring Composite Declaration

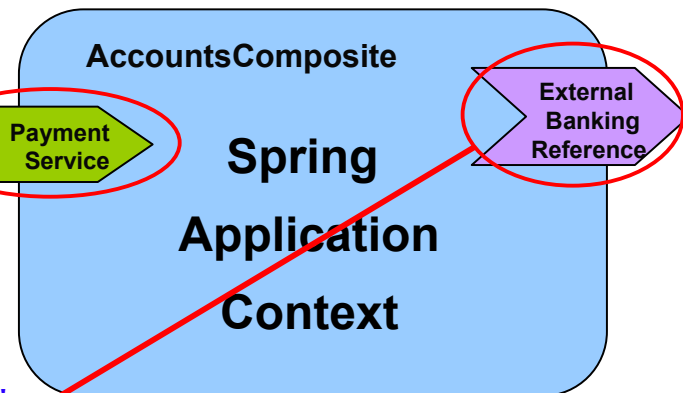
```
<component name="AccountsSystem">
  <implementation.spring
    location="AccountsSystemImpl.jar" />
  ...
</component>
```

```
<?xml version="1.0" encoding="UTF-8" ?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">
```

```
<bean id="PaymentService" class="com.foo.PaymentsBean">
  <property name="accountDataService" ref="AccountsLedgerService" />
  <property name="banking" ref="Banking" />
  <property name="currency" value="EURO" />
</bean>


<bean id="AccountsLedgerService" class="com.foo.AccountsLedgerBean" />
```

```
</beans>
```



Exporting Beans as SCA Services


```
<component name="AccountsSystem">  
  <implementation.spring  
    location="AccountsSystemImpl.jar"/>  
  
  <service name="PaymentService"/>  
  ...  
</component>
```



Expose the bean with this
name as a service

Injecting References to SCA Services

```
<component name="AccountsSystem">  
  <implementation.spring  
    location="AccountsSystemImpl.jar"/>  
  <service name="PaymentService"/>  
  
  <reference name="Banking"/>  
</component>
```



Bean made available in
a parent context

Architecting Spring and SCA Applications

- Spring and SCA are complementary
 - Use spring to build **implementations** of “coarse grained” service components
 - Bean implementations of “remotable” services
 - Invocations to referenced remote services
 - Local, fine grained components as spring beans
 - Bring in SCA to **expose services**, wire service components together, deal with **heterogeneous, distributed** systems
 - Handles configuration of bindings, endpoint
 - Deals with non-Java platform elements like BPEL, PHP

Architecting Spring and SCA Applications

- Spring and SCA are complementary
 - Enterprise services
 - SCA provides **policy and policy attachment** for security, reliability, transactions
 - Integrates with Spring security, transactions
 - Translates to WS-Policy, WS-Security, etc., for web services
 - Equivalents for other communication mechanisms
 - Distributed systems
 - SCA can describe solutions composed from service components running on **multiple, distributed runtimes**

Summary

- SCA provides a model for distributed composite service-based applications
- Spring provides a framework for building service components from simple Java objects
- SCA and Spring combine effectively to build distributed services applications

For More Information

- SCA-related sessions
 - **TS-8554**—Building, Assembling, and Deploying Composite Service Applications
 - **TS-41500**—Service Component Architecture Meets the Java Platform, Enterprise Edition (Java EE)
- SCA-related BOFs
 - **BOF-8238**—Building Composite Services Applications
- SCA
 - www.osoa.org
- Spring Framework
 - www.springframework.org



Q&A

Adrian Colyer

Mike Edwards



JavaOne

Spring and Service Component Architecture as the Basis for Distributed Services Applications

Adrian Colyer, CTO, Interface 21

Mike Edwards, Strategist, IBM

www.osoa.org

www.springframework.org

TS-8194